MURDOCH
UNIVERSITY
PERTH, WESTERN AUSTRALIA

School of Engineering and Information Technology

Dead Reckoning using an Accelerometer

Author: James Baker

Supervisor: Dr Gareth Lee

Semester 2, 2014

Executive Summary

The project aim was to develop a technology that would provide a way of navigating when GPS signals could not be used. This includes tunnels, inside buildings and in the city. The proposed solution was to use a dead reckoning (an out dated method of navigating) to navigate using an accelerometer to detecting movement.

The accelerometer acceleration samples would be double integrated to work out distance travelled in each direction. That is up and down, left and right, forward and backwards. A microcontroller would be used to control the device as well as an LCD screen and memory (EEPROM). The controller would be able to store the data to memory allowing it to imported into Excel after an experiment where the data can be manipulated.

In the end a prototype was build but failed to produce accurate calculations. The accelerometer values were hard to calibrate and therefore small errors would encroach on the data. This caused huge problems when integrating and seemed to magnify the error. The prototype would then insist that it has travel vast distances when in fact it has barely moved or not moved at all.

At the end of this document there are future recommendations. First of foremost, the calibration needs to be more accurate if the prototype to work. The two keys ways this would happen is if more samples could be taken to calibrate the accelerometer and/or the calibration value could have more decimal places.

Acknowledgments

Thank you to Dr Gareth Lee for giving up his valuable time to keep this project moving forward throughout the course of the semester. His advisements and time on the project were greatly appreciated.

For soldering the prototype to a circuit board, acknowledgment is given to Iafeta Laava. Additionally, his help in organising equipment and ordering parts for the project was greatly appreciated.

Contents

1 Introduction 1

1.1 Dead Reckoning 1

1.2 Inertial Navigation System 2

2 Design Strategy 3

2.1 Accelerometer 3

2.2 IMU 4

2.3 I2C 5

2.4 SPI 5

3 Selection of Devices 6

3.1 Accelerometer or IMU Selection 6

3.2 Controller Selection 8

3.3 Memory 8

3.4 LCD 9

4 Hardware Components 9

4.1 Accelerometer 9

4.2 Controller 10

4.3 Memory 11

4.4 LCD 12

4.5 Logic Level Converter 13

5 Building the Prototype 13

5.1 Address on the I2C bus 15

5.2 Hardware Problem 16

5.3 Software 21

5.4 Experiments 24

5.4.1 Experiment 1 25

5.4.2 Experiment 2 26

5.5 Results 27

5.5.1 Experiment 1 27

5.5.2 Experiment 2 33

5.6 Correction to Results 36

6 Conclusion 42

7 Future Recommendations 43

8 Bill of Parts 45

9 Appendix A: Software 46
9.1 Collecting Samples 46
9.2 Extracting Data 55
10 Appendix B: Calculations in Excel 61
References 62

List of Figures

Figure 1-1, Example of Dead Reckoning 1
Figure 2-1, Process of Prototype 3
Figure 4-1, Inertia Measurement Unit (IMU) 9
Figure 4-2, Arduino Due Microcontroller 10
Figure 4-3, 24LC256 EEPROM 12
Figure 4-4, LCD Screen 12
Figure 4-5, Logic Level Converter 13
Figure 5-1, Prototype on Bread Board 16
Figure 5-2, Prototype Soldered 17
Figure 5-3, Prototype SCL line 19
Figure 5-4, Prototype SCL line for 3.3V & 5V 20
Figure 5-5, Prototype SCL line for 3.3 & 5V 21
Figure 5-6, Flow Chart of first Sketch 23
Figure 5-7, Flow Chart of second Sketch 23
Figure 5-8, Nilfisk 24
Figure 5-9, Experiment 1 Prototype 26
Figure 5-10 Experiment 2 - Prototype on Nilfisk Front 26
Figure 5-11, Experiment 2 Side 27
Figure 5-12, Experiment 1 - X axis 28
Figure 5-13, Experiment 1 - Y axis 28
Figure 5-14, Experiment 1 - X & Y Samples 29
Figure 5-15, Experiment 1 - Velocity X axis 30
Figure 5-16, Experiment 1 - Velocity Y axis 30
Figure 5-17, Experiment 1 – Velocity 31
Figure 5-18, Experiment 1 - Distance travelled X direction 31
Figure 5-19, Experiment 1 - Distance travelled Y direction 32
Figure 5-20, Experiment 1 - Distance travelled 32
Figure 5-21, Experiment 2 –Attempt 1 - Velocity 33
Figure 5-22, Experiment 2 - Attempt 2 - Velocity 34
Figure 5-23, Experiment 2 - Attempt 3 – Velocity 34
Figure 5-24, Experiment 2 - Attempt 1 – Position 35
Figure 5-25, Experiment 2- Attempt 2 – Position 35
Figure 5-26, Experiment 2 - Attempt 3 - Position 36
Figure 5-27, Experiment 1 - X axis 37
Figure 5-28, Experiment 1 - Y axis 37
Figure 5-29, Experiment 1 – X axis 38
Figure 5-30, Experiment 1 – Y axis 38
Figure 5-31, Experiment 1 - X axis 39
Figure 5-32, Experiment 1 - Y axis 39
Figure 5-33, Experiment 1, X axis 40
Figure 5-34, Experiment 1, Y axis 41
Figure 5-35, Experiment 1 - X axis 41
Figure 5-36, Experiment 1 - Y axis 42

List of Tables

Table 3-1, Options for Accelerometers 7
Table 3-2, Options for IMUs 7
Table 3-3, Options for Controller 8
Table 5-1, Address of I2C Devices 15
Table 5-2, Experiment 1 - X axis 28
Table 5-3, Experiment 1 - Y axis 29
Table 5-4, Experiment 1 - X axis 40
Table 5-5, Experiment 1, Y axis 40
Table 7-1, Bill of Parts 45

List of Equations

Equation 1-1 2
Equation 10-1 61
Equation 10-2 61
Equation 10-3 61
Equation 10-4 61

Terminology and Acronyms

GPS	Global Positioning System
PCB	Printed Circuit Board
IMU	Inertia Measurement Unit
LCD	Liquid Crystal Display
PWM	Pulse Width Modulation
INS	Inertial Navigation Systems
MEMS	Micro Electro-Mechanical System
I2C (I^2C)	Inter Integrated Circuit
SDA	Serial Data Line
SCL	Serial Clock Line
TWI	Two Wire Interface
SPI	Serial Peripheral Interface
SSI	Synchronous Serial Interface
SCLK	Serial Clock
MOSI	Master Output Slave Input
MISO	Master Input Slave Output
SS	Slave Select
DMP	Digital Motion Processor
USB	Universal Serial Bus
DC	Direct Current
IDE	Integrated Development Environment
EEPROM	Electrical Erasable Programmable Read-Only Memory
LCD	Liquid Crystal Display
DIP	dual in-line package

1 Introduction

The purpose of this project is to develop a solution to navigate in areas where global positioning system (GPS) signals are unable to be used [1]. There are several places where GPS cannot be used such as in tunnels, within buildings, or in the city where large sky scrapers block and distort signals. Therefore there is a need to produce a technology that will enable navigation in these problematic areas.

It is proposed that using a Micro Electro-Mechanical System (MEMS) accelerometer would be used as a remedy for this problem [2]. It will sense movement and in turn be used for navigation. For this system to work, the accelerometer will need to be very accurate and very sensitive to detect movement to allow for position to be estimated. Additionally, it will need to take samples very quickly as if it takes too long to react, the object could have moved without the sensor noticing.

As this is a university project, it will need to be very cheap to develop, sticking to a rigid schedule whilst also being very accurate at navigating. The cost of the project can only be within a few hundreds of dollars. The schedule will incorporate a limited time frame of one semester and include deadlines of project plan, progress report and final presentation.

The way in which acceleration measurements will be used to figure out location is by a method known as dead reckoning [3]. The preceding section will provide a brief explanation what it is.

1.1 Dead Reckoning

Dead reckoning is a very old technology. It is a method used to determine location by using a previous position, velocity and time travelled. It has been used extensively in navigating ships and planes before the implementation of GPS. A simple and theoretical example is used below to help explain what dead reckoning is.

As depicted in Figure 1-1, Example of Dead Reckoning, a plane is to take off from position x (a known fixed position) and to travel to position y which is 1000 km west. The pilot is travelling at a constant speed of 500km/h west. One hour into the journey, the pilot wants to know his/her position. The pilot would use Equation 1-1 to figure out where he/she is. Whereby he/she would multiply the velocity and time to work out the plane's location, which would be 500 km/h multiplied by 1 hour. This would result in their location 500 km west of the starting positon x and therefore have another 500 km to go before reaching their destination.

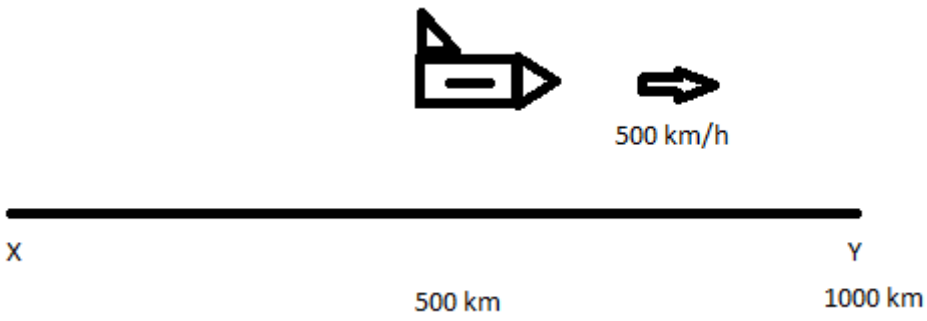

Figure 1-1, Example of Dead Reckoning

The basic formula for working out dead reckoning is [3]:

$$DR = Distance \times Time \quad \text{Equation 1-1}$$

The mention example is very basic but provides a good starting point of defining what dead reckoning is. That being said, the actual implementation is much more complicated. For instance, pilots would need to incorporate all three directions x, y and z. Furthermore, the wind direction and speed would also need to be considered as this would push them off course. However, if all these aspects are carefully accounted for, then dead reckoning can be an accurate way of navigating ships and planes.

Although at a first glance, this method may seem to be a robust method of navigation, it does have some flaws. Chief of which is the accumulative errors. When a measurement is taken, speed of a plane for example, there will always be some error associated with that measurement. As the measurement is then used in working out location, it will inherit the error. Furthermore, when working out the subsequent location, the new measurement (speed) will again have error. Therefore, the new position will contain the new error plus the error in the previous calculation. This process will continue and error will be growing. [3]

This is why it is called accumulative error as errors in working out position will always be increasing. It will reach a point where error will be so large that calculating the new location would be so far off target that it would render the calculation unusable and the method will fail. Working out when this has occurred will be unique to the specific application and will fundamentally depend on running time and how accurate the measurements have been taken. [3]

Although the accumulative errors impede the effectiveness of dead reckoning, it has been very successful in the past. For instance, in 1927 aviator Charles Linbergh travelled from the United Sates to France using dead reckoning as a method of navigating [3]. Therefore, if dead reckoning is used correctly, it can be very accurate for certain applications.

1.2 Inertial Navigation System

Dead reckoning has been made redundant by GPS as the main source of navigating ships and planes. However, inertial navigation systems (INS) are being increasingly used as supplementary system. They contain an on board controller to process the measurements, an accelerometer to measure acceleration and gyroscope for orientation.

The INS is a closed system that does not need outside information such as a GPS signal to workout location. It uses dead reckoning as a method of calculating the position and has to integrate acceleration of the object to convert to velocity. Then integrating velocity to establish distanced travelled.

Combining the GPS and INS data can reduce errors and if for whatever reason the GPS signals cannot be received, the INS will take over as the main tool of navigation.

INS suffers from integration drift. This is where the small errors in measuring acceleration are magnified when double integrating this measurement to work out velocity. The velocity is used to calculate position and will therefore be contain the large error. As the new position is based upon

the previous position, the new position will contain more error. Therefore to make INS accurate, corrections will need to periodically put the dead reckoning system back on course. [4]

2 Design Strategy

To test whether or not the dead reckoning system with an accelerometer is an adequate solution for navigation in areas where GPS does not work, a prototype would need to be built and tested. Initially, the prototype was thought to consist of an accelerometer that will infer object acceleration, a microcontroller to control the sensor, memory to store the data collected, and a LCD to display relevant information.

The fundamental process of the proposed prototype can be seen in Figure 2-1, Process of Prototype below. It is every basic but Illustrates how the system should work.

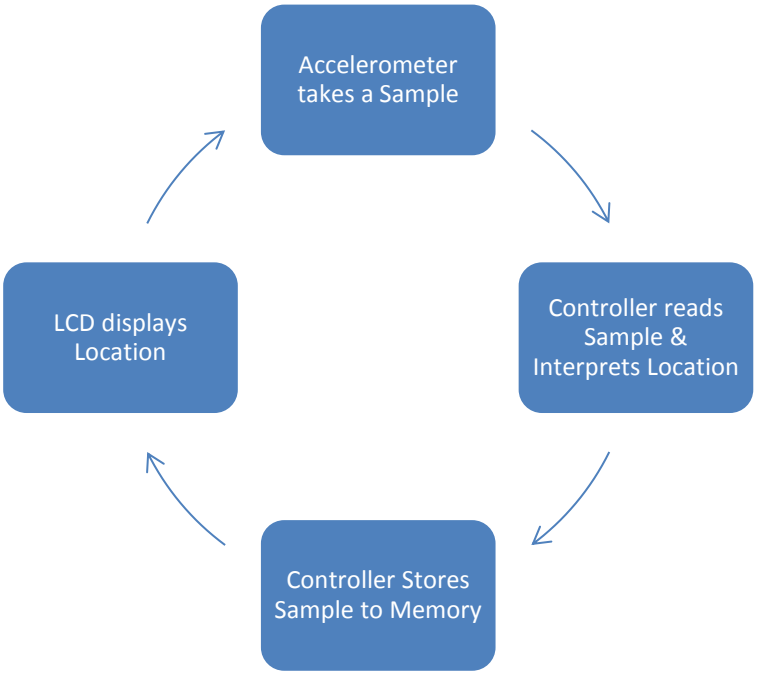

Figure 2-1, Process of Prototype

Before purchasing the components, some back ground research on the devices was conductor. A summary of relevant information is provided below.

2.1 Accelerometer

Accelerometers are made of micro electro-mechanical systems (MEMS) where tiny cantilever beams are used. When the object in question moves, the beams will bend and flex and this is in turn is used to calculate the object's acceleration. [5]

An unfortunate attribute of accelerometers is that they measure what is called proper acceleration. This is not the just the acceleration of the object but also includes the gravitational pull of the earth. Furthermore it behaves slight different to what is expected. For instance, when an accelerometer is in free fall, the accelerometer will measure 0 g-force on its axes. This is because the accelerometer structure is falling at the same rate as the beams inside the accelerometer are falling. The net result

is an output of 0 g-force instead of 9.8 m/s/s. However, knowing this information means it can be allowed for when using this device. [6]

There are two types of accelerometers on the market. One that outputs an analogue signal and the other has a digital interface. The analogue accelerometer outputs a pulse width modulation (PWM) where it frequencies will remain constant but its duty cycle will change in proportion to acceleration. [7]

On the other hand, accelerometers with a digital interface will commonly communicate using I2C, SPI or USB communications. That means this type of accelerometer has already taken the next step of converting acceleration into a language that micro controllers can understand. For this reason, accelerometers with a digital interface are much more common than their analogue counterpart. [7]

Although there is an abundance of accelerometers on the market that are cheap and very accurate, they are mostly unusable to this project. That is because the component size and pins are so small that wiring up a circuit by hand would be impossible. Therefore, an accelerometer on a breakout board needed to be purchased.

A break out board is simply a sensor, which is placed on a small PCB that allows the pins to be accessible by hand. They usually come with holes spaced at 0.1 inch spacing that header pins can be soldered to. This sizing also enables it to fit into the holes of a breadboard making it ideal for building prototypes.

Accelerometers often come with programmable measurement ranges. They usually have a minimum measurement range of ±2 g force and go up to measurement ranges of ±24 g force. However, selecting the high measurement range does not mean it will be the ideal choice for every task. This is because configuring the accelerometer to a larger measurement range reduces the precision of the accelerometer. Therefore, if it is more important for a task to be able measure vast measurement ranges, then the accelerometer would be configured to the highest measurement range. By contrast, if accuracy is more important and the accelerometer won't be measuring immense g forces, then the minimum measurement should be selected.

Understanding why this happens can be explained by the amount of bits available to provide a reading. For example, if the measuring range is between ±6 g then there is a total range of 12 g. This 12 g has to split up into 4096 (2^{12}) parts as that is the amount of different sequences are possible for 12 bits of binary. So the 12g are broken up by 4096 parts and subsequently, the when the last significant bit changes, the measurement will either increment or decrement by 3 mg (12/4096 = 3mg).

2.2 IMU

An alternative of using an accelerometer is an inertia measurement unit (IMU). Generally, IMUs have on board an accelerometer, magnetometer and gyroscope all in the one package. The magnetometer is a sensor for magnetic fields and has the potential of incorporating a compass to the project. A gyroscope is a sensor used for measuring rotation and tilting. So obviously using an IMU would provide the project with the ability of adding more features if chosen.

There are different types of IMUs and the way in which they are distinguished from one another is by their degrees of freedom. For instance, a device that has an accelerometer that measures in the x, y, and z axes provides three degrees of freedom. When this is combined with a gyroscope that also measures on the x, y, and z axis, it will provide an additional 3 degrees of freedom. If it were only these two sensors on the IMU the degrees of freedom would surmount to 6. If an additional magnetometer was also on the IMU and could measure in the x, y, and z axis, then again will provide an extra 3 degrees of freedom. In the end it will provide a total of 9 degrees of freedom.

It is worth noting here that if an IMU did say it had 9 axes, there is no rule or standard defining what those 9 axes are referring too. The IMU could have 9 accelerometers measuring acceleration in the same direction. So if an IMU says it has multiple axes, what sensors and what axes they are measuring on is unknown. Reading the description or data sheet of the device will need to be looked into [6].

2.3 I2C

I2C stands for inter integrated circuits. It is bus invented by Philips Semiconductors used for communicating at relatively low speed and short distances for peripheral devices. It is capable of hiving multiple masters, and multiple slaves. Each device is required to have a unique address on the bus [8].

The bus consists of a Serial Data Line (SDA) and a Serial Clock Line (SCL). Between the SDA and the voltage supply there is a pull up resistor and the size of the resistor determines the speed at which the bus operates. The value of the pull up resistor is sometimes specified in the data sheet and sometimes soldered into the circuit of the device.

There are fixed speeds at which the I2C bus can communicate. They are [8]:

- 100 kHz
- 400 kHz (fast mode)
- 1 MHz (fast mode plus)
- 3.4 MHz (high speed mode)
- 5 MHz (ultra-fast mode)

When investigating devices that communicate with I2C, it is important to look into the I2C communication speed capabilities. This is especially important if high speed is required.

Other companies have essentially copied I2C bus which they have renamed as Two Wire Interface (TWI). It was implemented by other vendors such as Atmel to due to get around trade mark issues [9].

2.4 SPI

Serial Peripheral Interface (SPI), otherwise known as Synchronous Serial Interface (SSI) is a synchronous serial data link that is full duplex. It is a standard that was named by Motorola and is commonly used for short distance communication. It may have only one master but can have multiple slaves. It is typically used for sensors, embedded systems and secure digital memory cards. [10]

SPI commonly has at least four wires. They are normally named as follows but may be called something similar [10]:

- Serial Clock (SCLK)
- Master Output, Slave Input (MOSI)
- Master Input, Slave Output (MISO)
- Slave Select (SS)

SS is a pin which when set low, indicates the device has been selected and the master will communicate with it. However, if there is more than one device on the bus, there will need to be one SS wire per slave. Otherwise, each device will not know when the master is trying to communicate with it. This is different to I2C where each device must have a unique address [10].

Intuitively, the devices on the bus are synchronised with the master's clock. The clock speed needs to be set at the speed no more than the maximum speed at which the devices can handle. The typical speed is generally a few megahertz [10].

3 Selection of Devices

The selection of components for the prototype was very important and could make or break of the project's success. So the method of selection of devices was to start at the most critical component first and then move to the next important component and so on until it reached the least critical part.

The most important component was thought to be the accelerometer. This is due to the fact that the entire project was based around the accelerometer's measurements. So the most accurate and quickest sample rate would provide the project with the best possible chance of success. It is why it is considered first in the component selection.

3.1 Accelerometer or IMU Selection

To figure out if an accelerometer or an IMU was to be used, a comparison between the two was made. Whichever would benefit the prototype the most would be selected. To start with, accelerometers were looked into first.

It was very important to purchase an accelerometer on a breakout board, because if only the accelerometer was purchased, the pins would be too small and not be accessible by hand. This greatly reduces the options leaving only three accelerometers to consider. The accelerometers are shown below in Table 3-1, Options for Accelerometers.

Considering that the project would be moving at walking speed, there is no need for vast measurement range. This means there is no point in analysing how accurate the accelerometer is on large accelerating ranges, what only needs to considered is how well it performs in the minimal measurement range. In this case, the ± 2 g and the ± 6 g ranges were investigated and the high measurement ranges were ignored.

The accelerometer that would benefit the prototype the most was thought to be the MMA8452Q as it was the most accurate (0.98 mg/LSB) when comparing to the LIS331HH and ADXL345. Also, it was

much cheaper. Unfortunate it only supports I2C. It would have been more beneficial to the project if it could have communicated with SPI as it is much faster. In any case, accuracy is of most importance and that is why the MMA8452Q was considered the best choose out of the accelerometers.

Accelerometer	Interface Type	I2C/SPI clock rate	Max Output data rate	Bits	Measurement range (g)	mg/LSB	Cost
LIS331HH	I2C/SPI	400KHz/ 10MHz	1000Hz	12	$\pm 6, \pm 12, \pm 24$	3, 6, 12	$27.95
MMA8452Q	I2C	400KHz/-	800Hz	12	$\pm 2, \pm 4, \pm 8$	0.98, 2, 3.9	$9.96
ADXL345	I2C/SPI	400KHz/ 5MHz	3200Hz	10	$\pm 2, \pm 4, \pm 8, \pm 16$	3.9, 7.8, 15.6, 31.2	$17.95

- *Measurement range – measure in g force*
- *mg/LSB – milli g force per last significant bit. How accurately the accelerometer can measure*
- *References* [11] [12], [13] [14], [15] [16]
- *Prices are in Australian dollars as of November 2014*

Table 3-1, Options for Accelerometers

It was also desired to select an inertia measurement unit (IMU) that would communicate using the SPI interface because it communicates at a much faster than I2C interface. SPI can communicate at a few Mbits/s whereas I2C on the market could only communicate at a maximum of 400kbits/s. However, a breakout board containing SPI interface could not be found and a decision was made to settle for a breakout board containing communication using I2C or USB. It was clear that there were three options available within the project's price range. These are shown below in Table 3-2, Options for IMUs.

IMU	Interface	DOF	Accelerometer measurement range	Accelerometer mg/LSB	Cost
Razor IMU	USB	9	$\pm 2, \pm 4, \pm 8, \pm 16$	3.9, 7.8, 15.6, 31.2	$74.95
MPU 9150	I2C	9	$\pm 2, \pm 4, \pm 8, \pm 16$	0.061, 0.122, 0.244, 0.488	$34.95
Minimu-9 V3	I2C	9	$\pm 2, \pm 4, \pm 6, \pm 8, \pm 16$	0.061, 0.122, 0.183, 0.244, 0.732	$47.75

- *Measurement range – measure in g force*
- *mg/LSB – milli g force per last significant bit. How accurately the accelerometer can measure*
- *References* [17] [16], [18] [19], [20] [21]
- *Prices are in Australian dollars as of November 2014*

Table 3-2, Options for IMUs

In Table 3-2, Options for IMUs, it can be seen that the MPU 9150 and the Minimu-9 V3 both have the same accuracy in the ranges of ± 2 and ± 4 g. The Razor IMU is not as accurate and was therefore ruled out of selection. The reason that the measurement range was

mentioned from ±2 and ±4g is because only walking speeds are will be measured. Therefore, the accuracy for greater measurement ranges is irrelevant.

To distinguish which of the remaining two IMUs is most suitable, the type of interface was look into. However, both communicated using I2C at a speed of 400kbits/s. So the ultimate way to differentiate between them was price. The MPU 9150 manufactured by Sparkfun was in the end considered to be the best choice.

When comparing the accelerometer to the IMU it is clear that the MPU 9150 had the advantage of potentially adding more features to the prototype. More importantly it was much more accurate than the MMA8452Q, as the LSB was 0.98 mg comparing that to the IMU of 0.061mg. The price of the MPU 9150 was also less expensive, at $34.95. As can be seen the MPU 9150 was the best possible choice for measuring acceleration and was chosen as the way of measuring acceleration for the prototype.

3.2 Controller Selection

There were two types of Arduino microcontrollers that were considered for the project; the Arduino Due and the Arduino Mega. They are very similar products however the clock speed at which they operate is quite different. The Due was much faster operating at 84MHz and the Mega operates at only 16MHz. Both controllers could have been supplied by Murdoch University so cost was not a factor. So it made sense that the Arduino Due would be the better choice as it is faster.

The only drawback in selecting the Arduino Due is that it operates at 3.3 V. Most of the peripheral devices obtainable operate on 5 V, like the LCD screen. So rather than turning back and selecting the Arduino Mega which operates on 5 V, a very cheap logic level converter was sourced. It allows devices that operate on different voltages to interface using I2C. The cost of the logic converter was low and will not affect the prototype in a negative way. Therefore, the logic level converter was ordered nullifying the drawback of the faster Arduino Due.

It should be noted here that it was not known exactly how much code would be needed for the controller to process. Therefore, due to the fact the one controller was much faster than the other, and there weren't any other obvious differences between the two, this made the decision easy to select the Arduino Due. Even if the controller had to be purchased, the cost between the two is so small and the Arduino Due would still have been chosen.

Controller	Clock Speed (MHz)	Interface capabilities	Language	Digital I/O	Cost
Arduino Due	84	SPI/I2C/USB	Wiring	54	$56.99
Arduino Mega 2560 R3	16	SPI/I2C/USB	Wiring	54	$52.43
References: [22] [23] [24]					

Table 3-3, Options for Controller

3.3 Memory

Memory is needed to store data that the sensors will produce. This will allow information to be looked at and analysed after an experiment has been conducted. Ideally, it should highlight where the prototype could be improved, be able show if there is a problem, or show the prototype is

working as intended. If there were no memory to store experimental data, an experiment will be conducted and there would be no way of analysing the measurements. The memory will provide a means to make informed decisions.

The memory chosen for the prototype was the I2C EEPROM - 256kbit because it was cheap, came from the same supplier as the other components and could communicate using I2C. The downside is that the memory does not communicate using SPI, so it would not be as flexible as the LCD that could communicate with both types of interfaces. The positive side is that will work using the 3.3 or 5 V. This means it can communicate using the Arduino voltage (3.3V) or off the LCD (5V) voltage.

3.4 LCD

To display relevant information about the prototype's values, a simple LCD was needed. As the other devices communicate using I2C, it was thought it would be best to obtain a LCD screen that would also interface using this technology, rather than making it harder and simultaneously communicating with several interfaces at once. It would also reduce wiring and make the circuit easier to follow if everything was on the same bus.

The "I2C/TWI LCD1602 Module" was chosen because it was cheap and could still communicate with both SPI and I2C interface. Therefore, if there were sufficient time during the project a comparison could be made between using SPI and I2C. The LCD would be able to be used in both circumstances.

4 Hardware Components

Descriptions of the core components chosen for the prototype are covered in this section. They are:

- Accelerometer
- Controller
- Memory
- LCD display

4.1 Accelerometer

The accelerometer used for measuring acceleration for this prototype is on an Inertia Measurement Unit (IMU) breakout board. The IMU is called MPU 9150 which has 9 degrees of freedom that has an accelerometer, gyroscope and magnetometer. All three of the devices are constructed out of MEMS technology making them cheap, accurate and very compact. They also measure their respected sensors on all three axis x, y and z. [25]

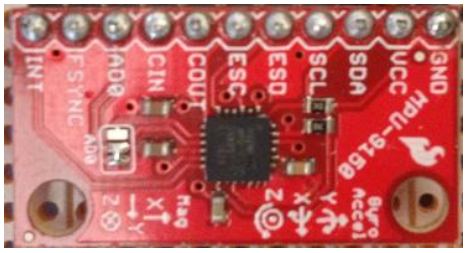

Figure 4-1, Inertia Measurement Unit (IMU)

The IMU interfaces with I2C at 100 kHz. This is relatively fast as the accelerometer samples at 1 kHz and the gyroscope samples at 8 kHz.

Key features of the Accelerometer include [19]:

- Programmable measurement ranges of: $\pm 2g, \pm 4g, \pm 8g, \pm 16g$
- ADC word length: 16 bit
- Sensitivity of: 16 384 LSB/g, 8 192 LSB/g, 4 096 LSB/g, 2 048 LSB/g
- Measure on three axis x, y, and z
- Output data rate of 1 kHz

Key features of the Gyroscope include [19]:

- Programmable measurement ranges of: $\pm 250\,^{\circ}/_{s}, \pm 500\,^{\circ}/_{s}, \pm 1\,000\,^{\circ}/_{s}, \pm 2\,000\,^{\circ}/_{s}$
- ADC word length: 16 bit
- Sensitivity of: 131 LSB/$\left(^{\circ}/_{s}\right)$, 65.5 LSB/$\left(^{\circ}/_{s}\right)$, 32.8 LSB/$\left(^{\circ}/_{s}\right)$, 16.4 LSB/$\left(^{\circ}/_{s}\right)$
- Measure on three axis x, y, and z
- Output data rate of 8 kHz

Key features of the Magnetometer include [25]:

- Measurement range: ± 12000 µT
- ADC word length: 13 bit
- Sensitivity: 0.315 µT/LSB

4.2 Controller

Arduino is a family of open source hardware microcontrollers with a free integrated development environment (IDE). They are designed to be a fairly simple to use and cost effective way of measuring the external environment through a variety of sensors. They allow any user to upload code to control, monitor or manipulate equipment and the environment. [26]

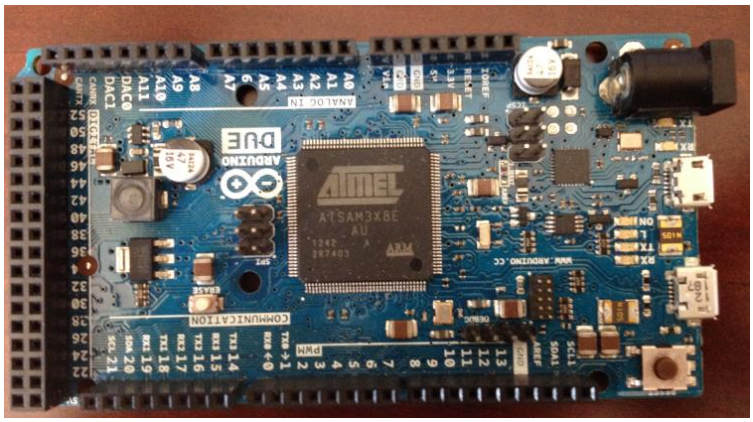

Figure 4-2, Arduino Due Microcontroller

The code is written in an implementation of wiring language which is similar to C/C++ [26]. It allows inexperienced people to write programs with relative ease who do not know the concepts such as classes, objects and pointers. However, people who do understand these concepts still can use them

[27]. The programs written are called sketches and are compiled in an environment known as the Integrated Development Environment (IDE).

The controller for this prototype is the Arduino Due. It is currently the most powerful Arduino microcontroller as it is the first to have the Atmel SAM3X8E ARM Cortex M3 CPU [28]. It is a 32 bit processor and has an 84 MHz clock. It also is the first to operate at 3.3 V as the others operate at 5 V.

As the Arduino Due operates at 3.3 V, it limits the range of products that can be connected straight to the inputs of the microcontroller. There are a lot of devices on the market that operate at 5 V as this this seems to be the industry norm. However new products appear to transitioning to operate on lower voltages such as 3.3 V. Obviously, care needs to be taken when selecting components, and if a device operates on a higher voltage than 3.3 V a step voltage converter or logic level converter will be required.

Summary of the key features of the microcontroller include [22]:

- Clock speed 84 MHz
- Operating voltage 3.3V
- 54 Inputs and Outputs
- 12 Analogue Inputs pins
- 2 Analogue Outputs pins
- 130mA Total DC Output current for all Inputs/Outputs
- 512 KB flash memory

The Arduino can be powered by a USB connection which provides 5 V at the programming port. This is convenient as it is needed to write sketches to microcontroller. The downside of this method of power supply is that it needs to be connected to the computer continuously. If it is desired for the Arduino to move around then the computer or laptop will be require to follow it. Furthermore, if the computer or laptop turns off then it will turn off the microcontroller as well. Fortunately the Arduino Due has a 2.1 mm power jack which is able to facilitate DC power supply. It is recommended to be supplied with a voltage range between 7 to 12 V. If it is supplied with 12 V or more than the on board voltage regulator may over heat or damage the board. On the other hand, if the voltage supply goes below 7 V then the 5 V power supply pin may drop to below 5 V. For this prototype, a 9 volt battery was used to supply power. [22]

Dedicate pins for the TWI bus are on pins 20 (SDA) and 21 (SCL), as well as pins SCL1 and SDA1.The wire library must be imported into a sketch for TWI to work on any of these pins. [29]

4.3 Memory

The memory chosen to store data produced by the prototype is the 24LC256 manufactured by Microchip Technology Inc. It is an electrical erasable programmable read-only memory (EEPROM) and is comes as a dual in-line package (DIP). It is capable of storing up to 32k bytes or 256k bits and interfaces using I2C. It operates in the ranges 1.8 to 5.5 V, allowing it to run straight off the Arduino Due microcontroller as it operates at 3.3 V. [30]

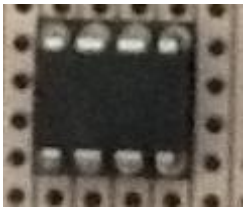

Figure 4-3, 24LC256 EEPROM

The device can be written to in two different ways. One way is by writing a single byte at a time and the other is by page write. Page write enables up to 64 bytes to be successively written. This is where the first byte is written to its designated storage location and the remaining 63 bytes are temporarily stored in the device's buffer. This is convenient if multiple bytes are required to be written to successive locations the EEPROM and would be quicker than writing one byte at a time. It will allow the bus to be free to do other tasks as soon as possible. [30]

Much as there are two ways of writing to the EEPROM, there are two ways in to read from the EEPROM. The first method is simply reading one byte at a time and the other is by sequential reading. Sequential reading is where the master does not send a stop bit after it receives a byte, instead it sends an acknowledge bit. This will let the EEPROM know that it is required to send the next byte. The internal pointer keeps track of what byte was sent and will move to the next byte. Once the master has read all the bytes it desires, it will send a stop signal at the end of the final byte read. [30]

4.4 LCD

The LCD chosen for this prototype is the Hitachi HD44780U LCD [31], and is packaged with I2C interface that has been put together by DFRobot. Unfortunately, the screen chosen for the prototype operates on 5 V. This is disappointing, as it is the only device chosen that interfaces with I2C using 5 V. To make the LCD compatible with the lower I2C voltage (3.3 V), a logic level converter was used.

The LCD is able to be programmed to display either 8 characters on one line or display 8 characters on two separate lines. It is able to display 208 5x8 dot character fonts and 32 5x10 dot character fonts, resulting in a total of 240 different character fonts.

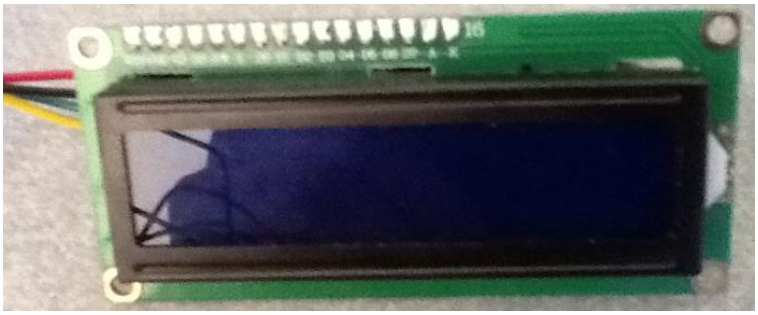

Figure 4-4, LCD Screen

4.5 Logic Level Converter

The logic level converter, sometimes referred to as logic level shifter, chosen for this prototype was bi-directional and manufactured by Sparkfun of a breakout board. It allows safe communication between 3.3 V and 5 V. It has four separate channels for communication allowing two different I2C communications as SDA and SCK will take up two channels.

Figure 4-5, Logic Level Converter

The logic level converter works due to n channel MOSFETs. They provide the way of switching of the signals. The two voltage levels are grounded on the same line but the high and low voltages are obviously kept separate.

To connect the logic level converter to the breakout board, headers were soldered to the board.

5 Building the Prototype

The prototype was built in a methodical and systematic fashion. Working with a single component at a time and making sure it worked before adding an additional device. Once it was believed that the component was working properly, a new component was added to the system. Then once both devices were working adequately, another device would be added to the prototype. This was the method use to complete the prototype.

To save time, example programs were uploaded to the microcontroller to get the peripheral devices working quickly. If a device did not work it would more than likely be an issue with the hardware as example software is usually written without fault. So instead of writing a sketch to an unfamiliar device as well as wiring it up this would provide two potential downfalls. Only the wiring was completed by the builder and code was left only. Therefore, there could only be an issue with the wiring because the code should have been written without fault by an experienced person. Implementing the example sketches removed the chance of code being wrong and should leave only the wiring to be incorrect. Ideally, this procedure would speed up the process of building the prototype.

The first part of building the prototype involved downloading the free software environment (IDE) from the Arduino website [32] . The version used was the 1.5.8 BETA which is for the Yun and Due boards. Once it was installed onto the Microsoft operating system, communication between the laptop and the microcontroller was made via the USB programming port. Once it was known that the laptop was successfully interfacing with the microcontroller, the next phase of building the prototype could begin.

The first component to be added to the Arduino Due was the LCD screen. The wiring was straight forward, including the logic level converter. However, once uploading the “HelloWorld” example

sketch, the LCD did not work. First, the wiring of the devices was thoroughly checked. The wiring instructions from the manufacturer of the logic level converter Sparkfun were consulted. It was very impressive and provided a lot of detail. From this information it was concluded that there could not possibly be an issue with the hardware and that this issue must lie in the "HelloWorld" example program. [33]

Searching for a resolution to the issue with the HelloWorld example program led to a scanning sketch. The I2C scanner sketch was downloaded from the Arduino website which - as it sounds - scans the I2C bus for devices. [http://playground.arduino.cc/Main/I2cScanner]. It was extremely helpful; it showed if something was connected to the I2C bus and displayed its address on the serial monitor. This works by the master (the Arduino controller in this case) sending out a "Wire.begin" message to address 1 to 126 on the I2C bus. It does this one at a time and immediately after the "Wire.begin" message has been sent it waits for a return. If there is a device on an address, it will return with a number and if it doesn't then there is nothing on the bus. If it returns 0 then a device with this address must have sent an acknowledgement on the SDA line back to the master. Therefore if a master sends out the "Wire.begin" message with an address, and it receives an acknowledgment, then there must be a device with that corresponding address on the I2C bus.

The I2C scanner was very helpful as it was used to figure out what the issue was with the LCD screen. The LCD screen did not work straight away after uploading the HelloWorld example program. Once the I2C scanner sketch was uploaded it was seen that the address being used in the Hello World example was wrong. This is because the Little Bird Electronics online retailer (were the device was bought from) said that the I2C address was 0x20 but it was in fact 0x27. If it were not for the I2C scanner, it would have taken a lot longer to figure out the cause of the problem. Nevertheless, once the address was adjusted in the HelloWorld example code, the LCD worked immediately. [34]

The next step was to get the memory device to work in conjunction with the LCD screen. It was wired into the current circuit according to the 24LC256 data sheet. This was very easy to implement and there were no issues to report. Programs were sought on YouTube as there were not any example programs for the EEPROM. Once a sketch was made and uploaded to the device, everything worked perfectly.

The final part to be added was the IMU. It was thought that it would be simply added to the current circuit like the EEPROM and everything would work accordingly. This could not be further from the truth. It never worked when adding the IMU to the established circuit. Initially it was unknown why it did not work at all and all the obvious things were checked making sure wiring was connected properly and that the addresses were correct. However this did not solve the problem and further investigation into the matter was needed.

After a period of time, a different approach was adopted. Instead, the IMU was connected to the microcontroller in isolation to the LCD and EEPROM. When running the I2C scanning sketch, the IMU address appeared on the serial monitor for the first time. Its address was 0x68 as expected as the A0 pin on the IMU breakout board was grounded. There is no other device on the I2C bus with the same address so there could not be an issues with addressing. The wiring was also constructed in the same way as before. Following the I2C scanner sketch, the example program MPU9150_raw was uploaded and was giving acceptable readings of acceleration, so there seemed to no issues with the device.

The conclusion was made that there must be some conflict with either the EEPROM or the LCD screen.

To figure out which device or devices was causing the IMU not to respond, the LCD and EEPROM were added one at a time while the IMU was connected to the I2C bus. Then it would be checked to see if both the devices respond to the master and behave as they were designed to. Firstly, the EEPROM was connected to the I2C bus which still had only the IMU connected to it. When running the I2C scanner sketch, they both responded successfully on the serial monitor. So there did not appear to be any problems with these two devices being on the bus at the same time.

On the other hand, when the LCD was added to the I2C bus, it caused a problem. When running the I2C scanner sketch, the IMU does not appear on the serial monitor, however the LCD and the EEPROM did. When the LCD was removed from the bus, the IMU address appeared straight away on the serial monitor, so the IMU must have sent back an acknowledgement. When the LCD was connected to the bus again, the IMU would not appear on the serial monitor. Cleary, the IMU would not work when the LCD was connected.

Next, the memory was removed so the LCD and IMU were the only devices connected to the bus. The I2C scanner was uploaded once again and only the LCD address would appear on the serial monitor. The LCD was removed and the IMU would then appear on the serial monitor. The LCD was then again put back into the circuit and then the IMU would not respond to the I2C scanner. It would also not work when the example program MPU9150_raw was uploaded to the microcontroller. Therefore, it was concluded that IMU would not work if the LCD was connected to the bus. The EEPROM and the LCD would always work regardless of what devices were added to the bus.

In the end it was established that the IMU was more susceptible to stray capacitance on the I2C bus then the LCD and EEPROM. They can still decipher what is a high and low signal is but the IMU cannot. The IMU must need a higher voltage for the high bit to be registered. So the capacitance must slow the transition down too much for it to reach an acceptable voltage. In the end the circuit was soldered onto a circuit board as well as including another resistor to the SCL and SDA line. This reduces the overall pull up resistance to compensate for the higher capacitance. This meant that SCL and SDA lines could reach a high voltage more quickly when transitioning. The drawback is that it will use more power.

A momentary contact switch was needed for the project to activate the programs and to stop it. This gave the operator the control to commence and stop a sketch from running. It was wired in accordance to the Arduino recommendations. [35]

5.1 Address on the I2C bus

The I2C addresses for the project components are shown in Table 5-1, Address of I2C Devices, below. They do have the capacity to be changed but only to ones defined in their data sheets.

Component	Address in Hex
IMU	68
Memory	50
LCD	27

Table 5-1, Address of I2C Devices

5.2 Hardware Problem

The only major problem with the project was with the IMU. It did not work if the LCD was connected to the prototype's circuit. It was believed that there is too much capacitance on the bus when the LCD is connected and the IMU cannot interface properly. It is believed that the SDA line cannot go from low to high quick enough for it to register that the master (Arduino Due) is trying to interface with it. The maximum allowable capacitance on the bus is 400 pF [8].

It should be pointed out here that the IMU briefly did work twice while the LCD was connected to the circuit. So there was some hope that the prototype would work with all devices and that there was just something wrong with circuit or program.

It was thought that it would be a good idea to measure the capacitance of the circuit to see if it was in fact the cause of the problem. This could not be done as capacitance of the circuit would need to be measured in Pico farads, which is extremely small. Murdoch University could only supply a multimeter to measure capacitance down to nano Farads. This would not provide sufficient evidence of high capacitance on the I2C bus. As an alternative, the circuit was investigated with an oscilloscope.

The oscilloscope probe was connected to the SCL line and the grounding probe connected to the ground of the circuit. When the IMU and EEPROM were the only two devices were connected, the oscilloscope showed a near perfect square wave. There was no evidence of high capacitance slowing the transition time between high and low. However once the LCD was added to the circuit, the oscilloscope showed that the SCL was constantly high.

Below in Figure 5-1, Prototype on Bread Board, show the prototype configuration and current status when the oscilloscope was connected the prototype.

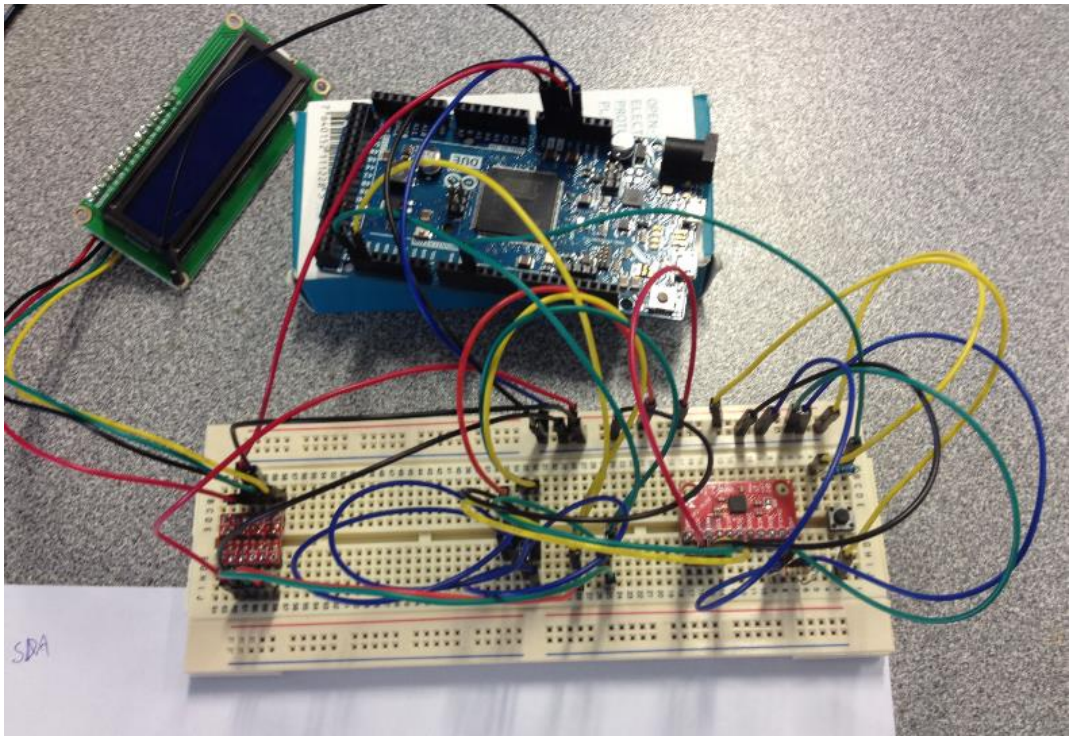

Figure 5-1, Prototype on Bread Board

It was then thought that there may be a wire or wires not being connected adequately to the bread board or that internally in the bread board there could be some wires crossing over. This would explain why it did briefly worked before and why it primarily doesn't work. To rule out the possibility of it being a wiring fault, it was thought that the circuit could be transferred over to a different bread board that is known to be sound. However, after some advice, it was decided to solder the circuit to a board. This would provide the additional advantage of reducing capacitance to the circuit as well as easily inspecting all the wires connected.

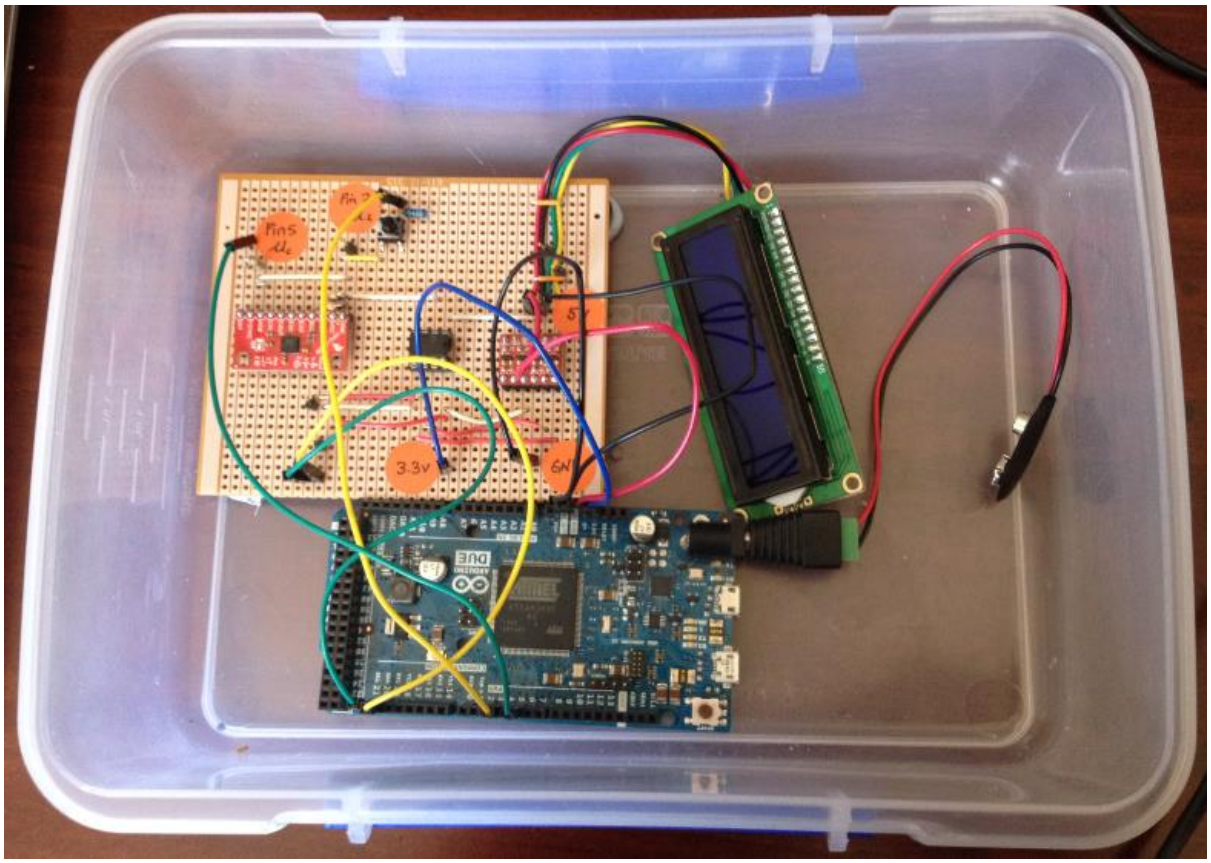

Figure 5-2, Prototype Soldered

In Figure 5-2, Prototype Soldered shows the prototype after it was soldered together. It is much neater and is contained in a plastic box.

After it had been soldered together on a circuit board, everything began to work as expect. Finally, it was thought that the issues with the I2C bus were over and that the problem before was caused by either a loose wire or something internally wrong with the bread board. However, after using the prototype for a period of time, it would without warning stop working. After unplugging and then plugging the USB cable back in to the laptop, it worked again for an extended period of time. Later on, after writing more code, it happened again. It was rare that the prototype would not work but clearly there was still an intermittent problem.

The idea that it was a loose wire or a problem with the internal parts of the bread board was abandoned. It was though once again that there was an issue with too much capacitance on I2C bus lines. This is because the new circuit should have less capacitance as bread boards impose more capacitance because of all there rungs of wire rung close together. The prototype worked most of the time rather than barely working at all. It was also thought that it could not be the wiring or the sketch as it was the same as before. So the only thing that should have been affected from transferring from the bread board to circuit board was the capacitance.

To prove that it was indeed too much capacitance on the I2C circuit, it should be able to be seen on an oscilloscope. The transition of the SDA and SCL lines from low to high should take a while and curve up instead almost seamlessly jumping from one state to the other. If the SCL line showed a perfectly square wave then it could not be the capacitance being too high and something else was causing the problem.

When the prototype had all components connected and was running a sketch, the probe of the oscilloscope was connected to the SCL line. As soon as the probe was connected to the SCL line, the prototype stopped working and the oscilloscope showed that the SCL line was continuously high (3.4 volts). When the probe was removed and the program restarted, it would work again. Then immediately adding the probe again to the SCL line, the program would stop running and the SCL line would continuously remain high. It was obvious that probe was causing some issue to the circuit.

It was later noticed that the probe was in the 1X position. This would impose an additional 95 pF onto to the bus. As it was assumed that the capacitance of the circuit was already high, so the addition of almost 25% of the allowable 400 pF added to the bus was thought to be the problem. To get around this issue, the probe was then selected to the 10X position which reduced capacitance of the probe to 16 pF, significantly less than before. The oscilloscope attenuation was then adjusted to 10X and the probe was then reconnected to the circuit. It worked instantaneously.

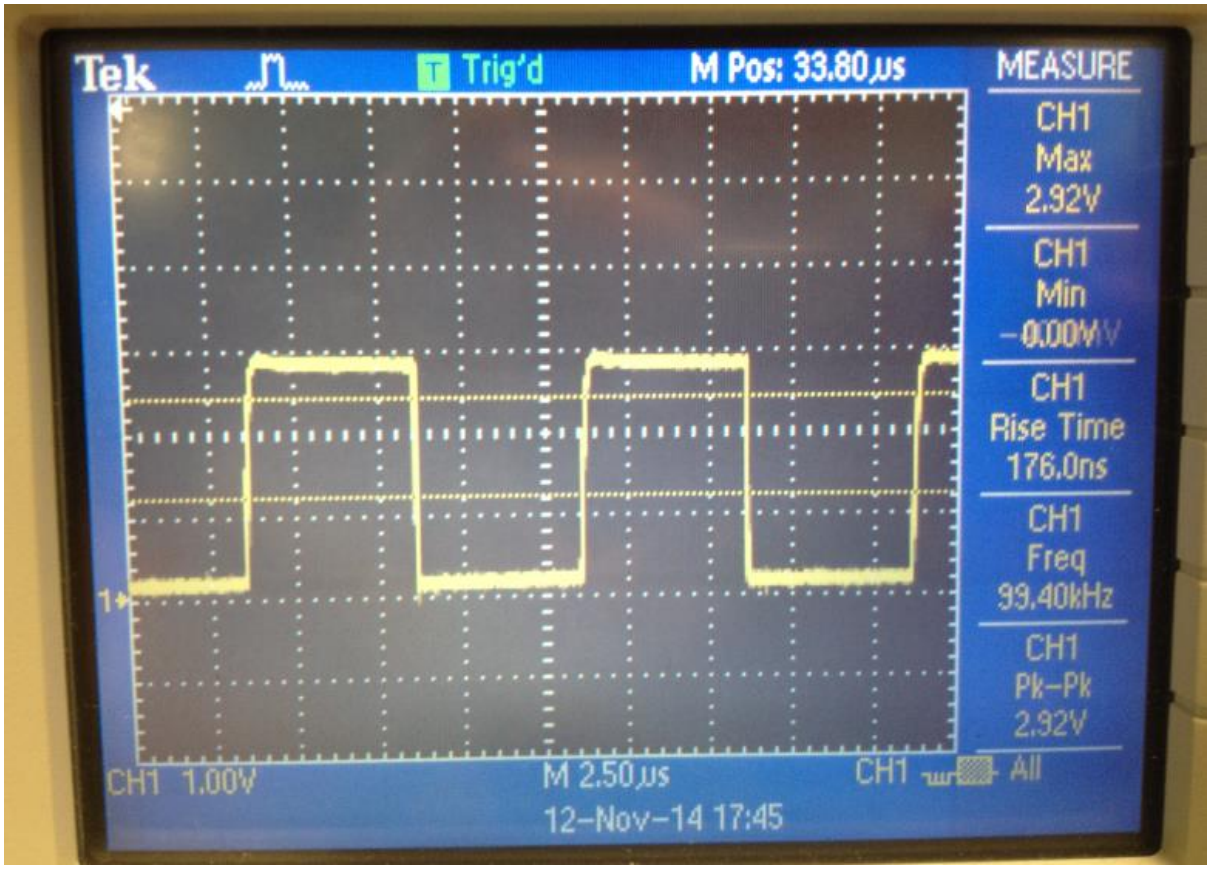

Figure 5-3, Prototype SCL line

The yellow line in Figure 5-3, Prototype SCL line*, is the SCL for the 3.3 voltage side of the prototypes circuit.*

After monitoring the SCL line, it was apparent that the low signals or bits were always very low and wouldn't go close to its maximum of roughly one volt to be recognised as a low bit. The high signal on the other hand did occasionally go close to reaching its minimum of 2.38 V to be recognised as a high bit. Therefore, it is believed that the reason for the prototype's rare failure was that it did not produce the high signals properly. The result would be that the interfacing would stop and the SCL lines would continuously stay high and the program would stop running.

The prototype would need to be robust to carry out good experiments so this issue needed to be rectified. Research on the matter suggested that pull up resistor could be reduce to compensate for high capacitance circuits. This would reduce the time take to achieve a high signal. It would also increase the high signal when viewing the SCL line on the oscilloscope

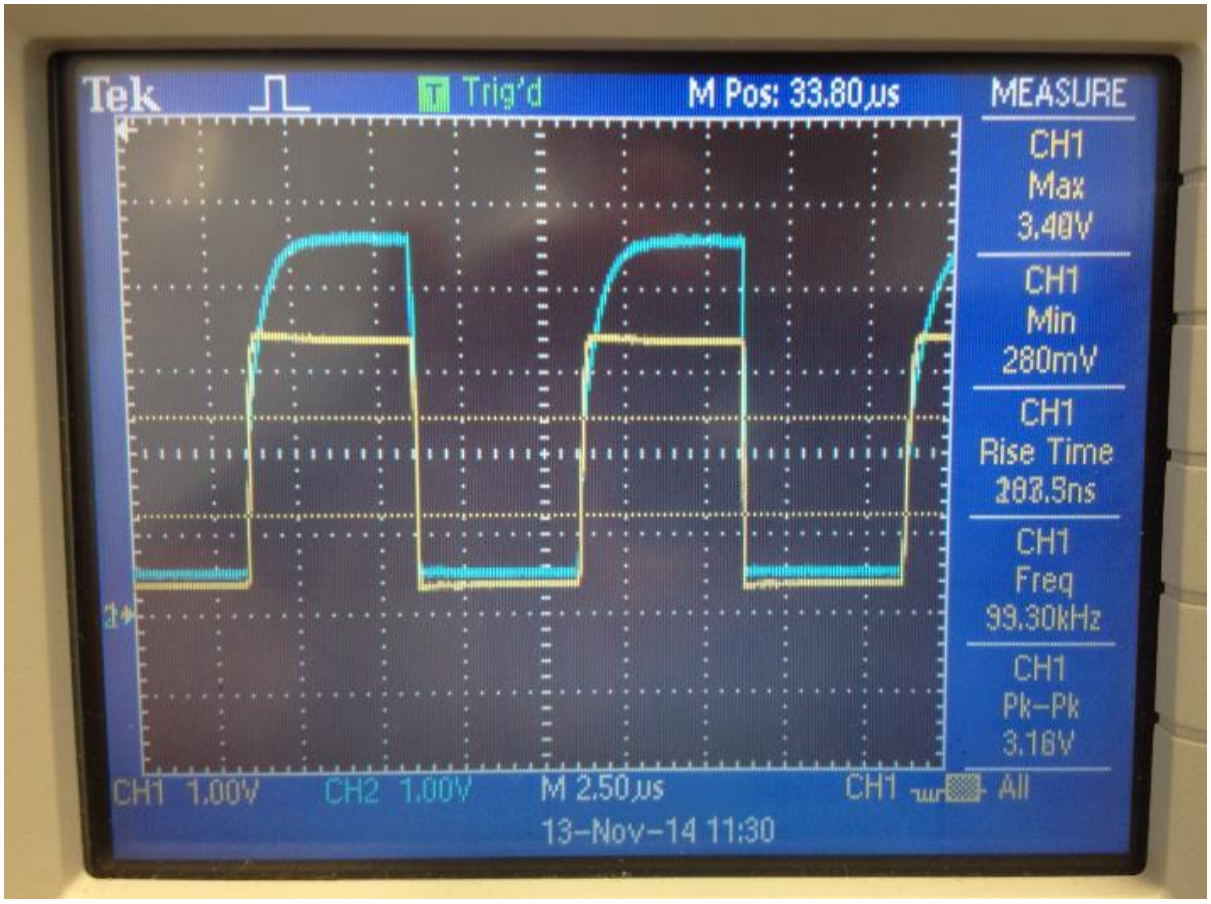

Figure 5-4, Prototype SCL line for 3.3V & 5V

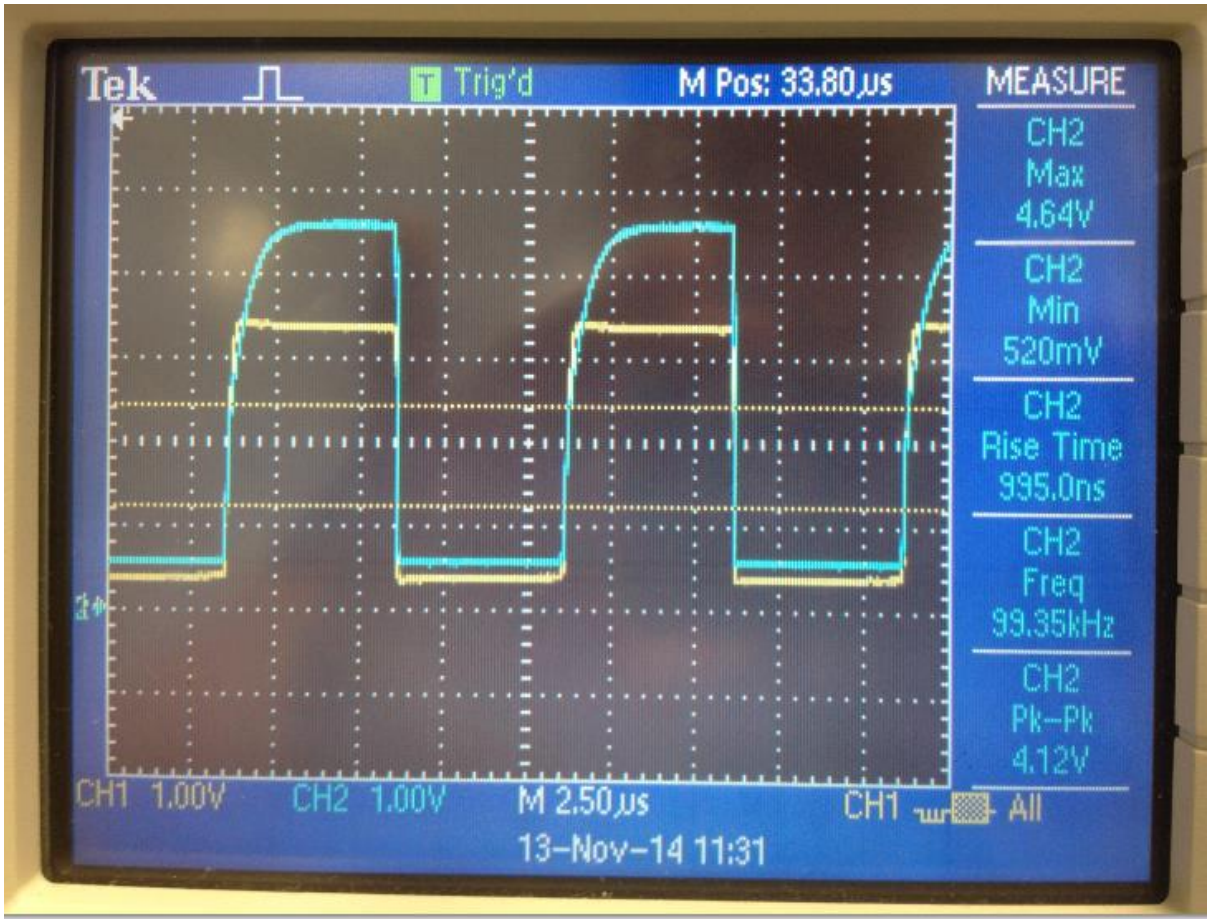

Figure 5-5, Prototype SCL line for 3.3 & 5V

As shown above in Figure 5-4, Prototype SCL line for 3.3V & 5V and Figure 5-5, Prototype SCL line for 3.3 & 5V, shows the SCL line on the oscilloscope. The yellow SCL on the 3.3 V side represents a very good square wave. On the other hand, the blue line in both figures shows the SCL line on the 5V side. It appears to suffer from induced capacitance as can be seen from the curving upwards to its maximum voltage. This could be caused by the wires connecting the LCD screen to the Logic Level convert being very close together. In any case, it doesn't affect the prototype in any negative way.

Since the implementation of the reducing the pull up resistance the prototype has been working very robustly and is yet to suddenly stop working. It has become very reliable and as far as the hardware concerned is adequate for the task of dead reckoning.

5.3 Software

As mentioned before in previous sections, example programs were used to quickly test and get components working quickly. They also were used to provide a starting point for developing on the prototype sketches. The example programs were incorporated in the additional libraries. Apart from the required header file and cpp file, they were also included.

The libraries included for the prototype were:

- I2C_dev [36]
- LiquidCrystal_I2C [37]
- MPU650 [38]
- MPU-9150_Breakout-master [38]

Example programs used were:

- HelloWorld from the LiquidCrystal_I2C file [37]
- MPU9150_raw form the MPU650 file [38]

Obtaining the data collected from the Arduino required a separate program as the IDE could not save a file straight to a text file or Excel spread sheet. It was recommended by the supervisor to use putty as a tool to save the data to a text and then import the text into a spread sheet.

The program to conduct experiments was split into two parts. The first part involved collecting the data from the IMU and storing the information into the EEPROM. The second sketch was written to retrieve the data stored on the EEPROM and load it into Microsoft Excel To achieve this, putty would be required to copy the data that was sent to the serial monitor and save it to a text file. The text file would then be imported into an Excel program.

A flow cart of how the first sketch works is shown below in Figure 5-6, Flow Chart of first Sketch. Before it goes into the cycle, it waits for the momentary switch to be pressed. Then, when it has been pressed it adds up all the samples collected in 5 seconds and then divides the value by the number of samples. This is used later to calibrate the IMU as it will not be perfectly flat so the x and y axis will be getting slightly influenced by the earth gravitational pull.

The sample rate at which the Arduino reads from the IMU is 50Hz. As this is too fast for the LCD screen to keep up with - let alone the operator - the LCD screen only gets updated at 2Hz. When the operator has finished his experiment, he would then push the momentary switch to end the experiment. The Arduino would then send a series of binary ones to be stored in memory to high light the point of when the experiment was finished.

The accelerometer never repeats the same number twice in any experiment conducted. This is because there is noise in the acceleration samples. Therefore, three binaries ones are saved to memory. This clearly defines the point of when the experiment has finished.

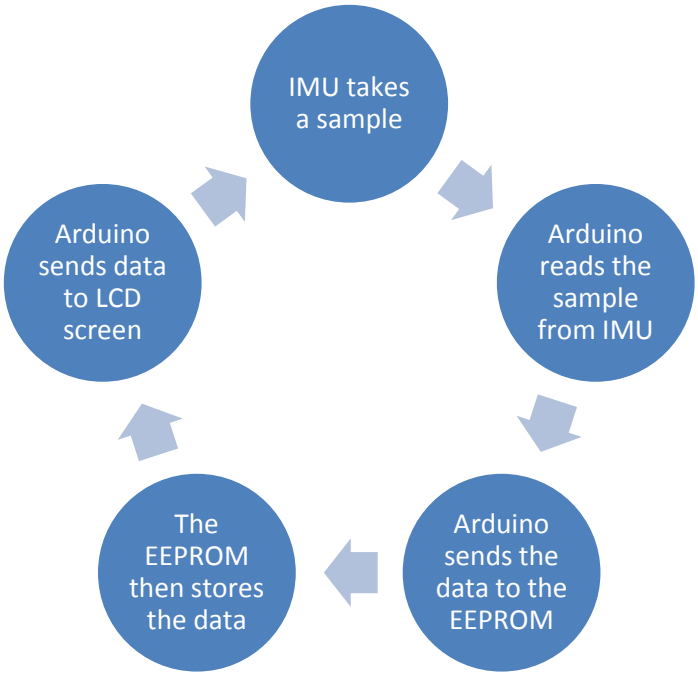

Figure 5-6, Flow Chart of first Sketch

The second program sequences are shown below in Figure 5-7, Flow Chart of second Sketch. First the program will be waiting for the momentary switch to be pushed before it will begin. When it has been pushed, the Arduino will begin reading data off the EEPROM. The data will be read as single bytes and then is joined together to form a signed 16 bit number and then written to the USB serial port. It will store the data to a text file.

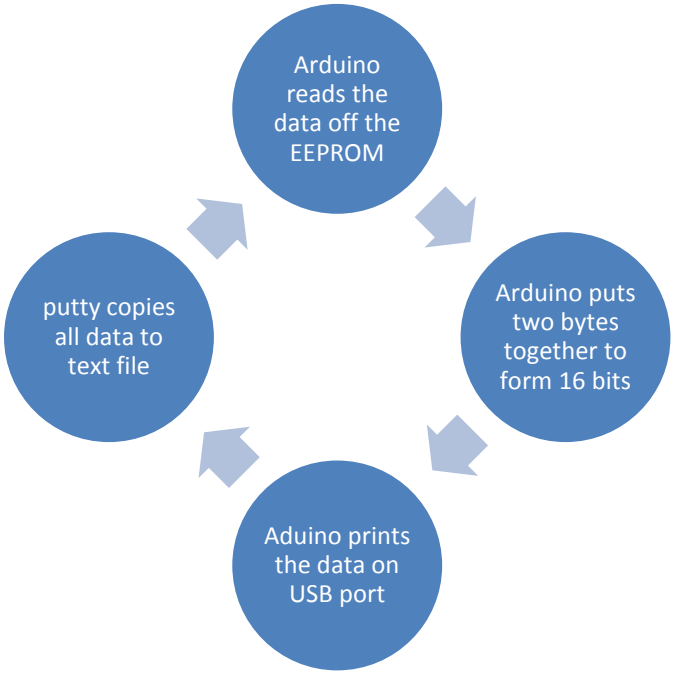

Figure 5-7, Flow Chart of second Sketch

When all the data has been retrieved, it would then be imported into an Excel spread sheet to be analysed.

5.4 Experiments

Experiments were conducted in an enclosed basketball court at Kwinana Recquatic. This place was chosen because it fits the criteria of being indoors (in a place where GPS signals would be hard to obtain) and it had a flat and predictable surface. Additionally, it was easy to get access.

To give the most consistent measurements, the prototype was fixed on top of a Nilfisk. This is a scrubber/dryer machine used for cleaning the basketball courts at Kwinana Recquatic. It was chosen because it is self-propelled when the accelerator is pushed in. This is ideal because it will accelerate up to its top speed and then hold that constant speed. It will be able to do this over and over again until it needs to be recharged.

Figure 5-8, Nilfisk

The velocity of the Nilfisk, shown above in Figure 5-8, Nilfisk, can be varied by turning a knob on the front of the machine. It does not show the speed it is traveling at on the LCD but velocity varies from approximately $1\ ms^{-1}$ to $4\ ms^{-1}$. It also has speed control; this was noticed as going down steep slopes, the machine would begin to increase velocity. It would take only a moment before it realises the increase velocity and then will slow its self-down to its original velocity. This behaviour is much like the cruise control on a motor vehicle but is not as refined.

Figure 5-9, Nilfisk front view

The Nilfisk has little bit of lag once the accelerator has been pressed before the machine begins to move. Once moving, it does jiggle about a little but it is not that noticeable. The operator of the machine has to continuously correct the path of the cleaning device as it will endlessly drift of course. Additionally when the accelerator is released, it is like there is an automatic break has been implemented. The Nilfisk does carry a lot of inertia as it is a heavy machine, so when releasing the accelerator, it does carry on forward for a brief movement before coming to a complete stop.

The circuit board was stuck to the inside of a clear plastic container with blue tack. The Arduino Due and the LCD screen were also in the container but were not securely fixed to anything. This enabled easy access to the Arduino to upload the two different sketches for the experiment. The container was then fixed to the Nilfisk with lots of blue tack. The container could very marginally wiggle on top of the Nilfisk but for the sake of the experiment it sufficed.

The basketball courts were vacant during the experiments and there was no interference. This gave the best possible chance of collecting good data to be analysed at a later date in Excel.

5.4.1 Experiment 1

The first experiment was the only experiment not to be carried out at Kwinana Recquatic. It was completed in the power laboratory at Murdoch University as seen in Figure 5-10, Experiment 1 Prototype. The prototype was operating stationary on top of a bench in the laboratory and its purpose was to see what the data would look like if it was static. Would the data suggest that it stayed in one spot or would it infer that it was moving around? During the course of this experiment, no one was using the bench or the bench adjacent to it. This would prevent any knocks to the table to influence the results.

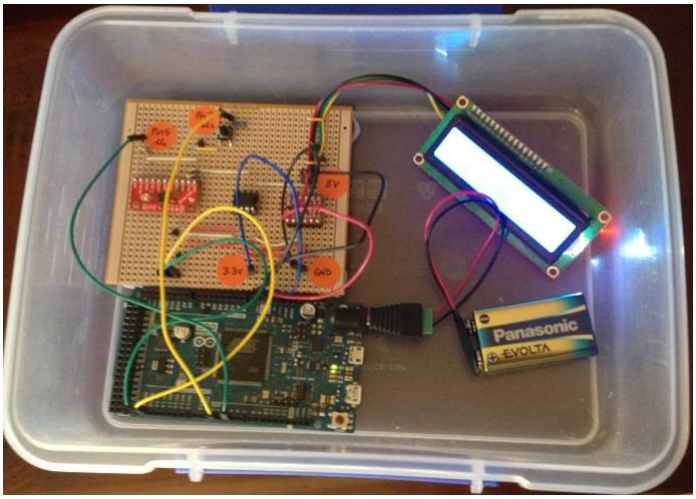

Figure 5-10, Experiment 1 Prototype

5.4.2 Experiment 2

Following the stationary test, the Nilfisk did straight line tests. The purpose of this experiment is to see how accurately the data would be in straight line movements. One of the perimeter lines of the basketball court was used to provide distance as well as help guide the operator in keeping the Nilfisk moving in a straight line. The white vacuum component at the back of the device was used as reference point. It would be lined up to the outside of the white line to start the experiment. At the end of the experiment, the user had to try and stop on the outside part of the white line at the far end of the basketball court. This was tricky, as the Nilfisk has a delay of when the accelerator has been taken off. It also carries a fair amount of inertia and will continue to keep moving forward for a small period of time. Therefore, a measurement for how far it exceeds the line was recorded and factored into the results.

The known distance the prototype travelled was 28.126m. Figure 5-11 Experiment 2 - Prototype on Nilfisk Front and Figure 5-12, Experiment 2 Side below shows the starting point of the experiment.

Figure 5-11 Experiment 2 - Prototype on Nilfisk Front

Figure 5-12, Experiment 2 Side

5.5 Results

The results for each experiment are shown below under their appropriate heading. When this document states the X and Y axis with capital letters, it means that this document is referring to the IMUs X and Y axis. If the lower case of x and y is used then that refers to the axis of the graphs, not the IMUs axis.

As the accelerometer measures proper acceleration, the Y axis of the IMU will measure the acceleration forwards and backwards of the Nilfisk. Positive values will represent forward acceleration and negative values will represent backward acceleration. The X axis of the IMU will measure the left and right movements. Negative values refer to left acceleration and right accelerations are positive values.

5.5.1 Experiment 1

The raw data collected had to be calibrated as the IMU did not sit perfectly flat. The influence of gravity was affecting both the Y and X axes. This had to be removed so only the actual acceleration of the prototype would be left. Therefore it was calibrated. The calibration was completed by sampling for 5 seconds at a rate of 50 Hz. The calibration value for the X axis was -1571 and the Y axis was 169.

The samples of acceleration appear to be accurate as can be seen below in Figure 5-13, Experiment 1 - X axis and Figure 5-14, Experiment 1 - Y axis. The data shows that the samples hover around the 0 point on the y axis. It can also be seen that the closer to the 0 mark, the more acceleration readings there are. The further away from the 0 of the y axis, the more spread out the data becomes.

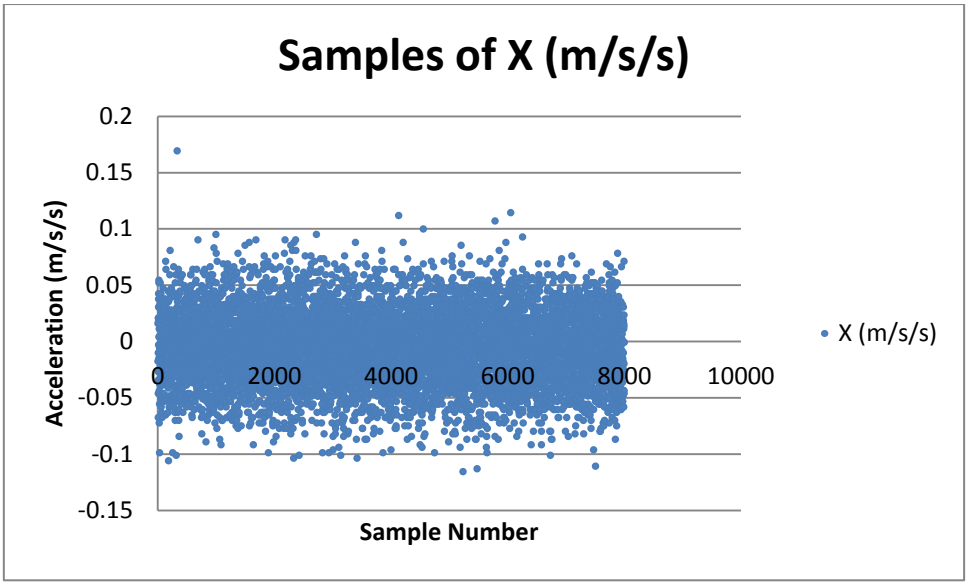

Figure 5-13, Experiment 1 - X axis

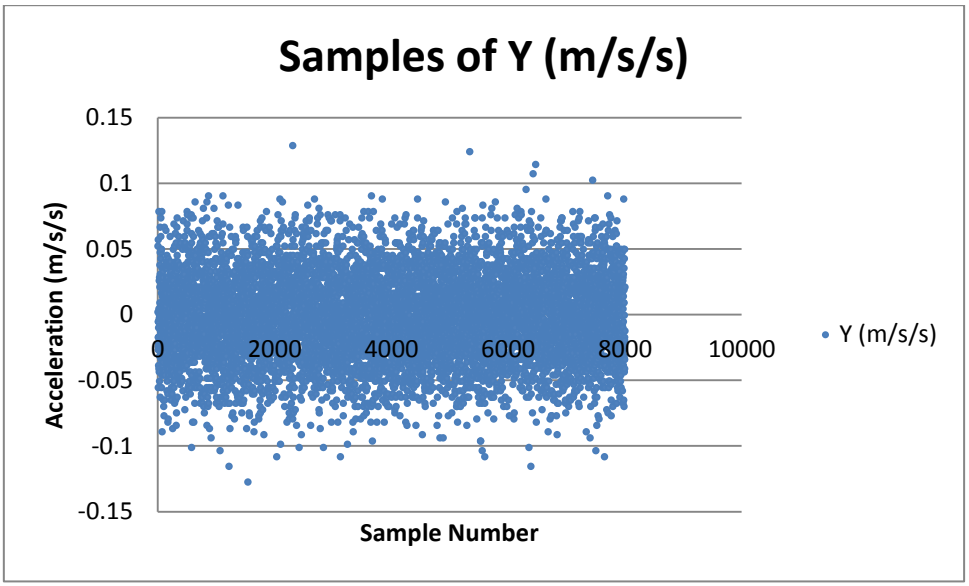

Figure 5-14, Experiment 1 - Y axis

Some of the key points of Figure 5-13, Experiment 1 - X axis can be seen below in Table 5-2, Experiment 1 - X axis. The calibration at the start seems to work at as the average is of the entire experiment was to the 6 thousandth value for the X samples. The standard deviation says that approximately 68% of the data lies between the -0.0317 and 0.0317 for normal distribution.

Total Samples of X (ms^{-2})	
Maximum X value:	0.169389767
Minimum X value:	-0.11552023
Average X value:	-0.00635553
Standard Deviation:	0.031693811

Table 5-2, Experiment 1 - X axis

Some of the key points of Figure 5-14, Experiment 1 - Y axis can be seen below in Table 5-3, Experiment 1 - Y axis. The calibration at the start seems to have worked at as the average is very close to 0. It is roughly -0.002 which is extremely close and is slightly more accurate than the X axis data. On the other hand the standard deviation is slightly more spread out as about 68% of the data lie between the -0.0322 and 0.0322 when comparing to the X samples. However, in the end the X and Y values are very closely matched as they should be.

Total Samples of Y (ms^{-2})	
Maximum X value:	0.128688339
Minimum X value:	-0.12749124
Average X value:	-0.00224657
Standard Deviation:	0.032218587

Table 5-3, Experiment 1 - Y axis

When placing both the X and Y values from the IMU onto the same graph, they also show that the points hang around the 0 mark of the x and y axis. The Y values from the IMU are placed on the y axis of the graph and the X values from the IMU are placed on the x axis. Figure 5-15, Experiment 1 - X & Y Samples, shows the comparison of the X and Y values.

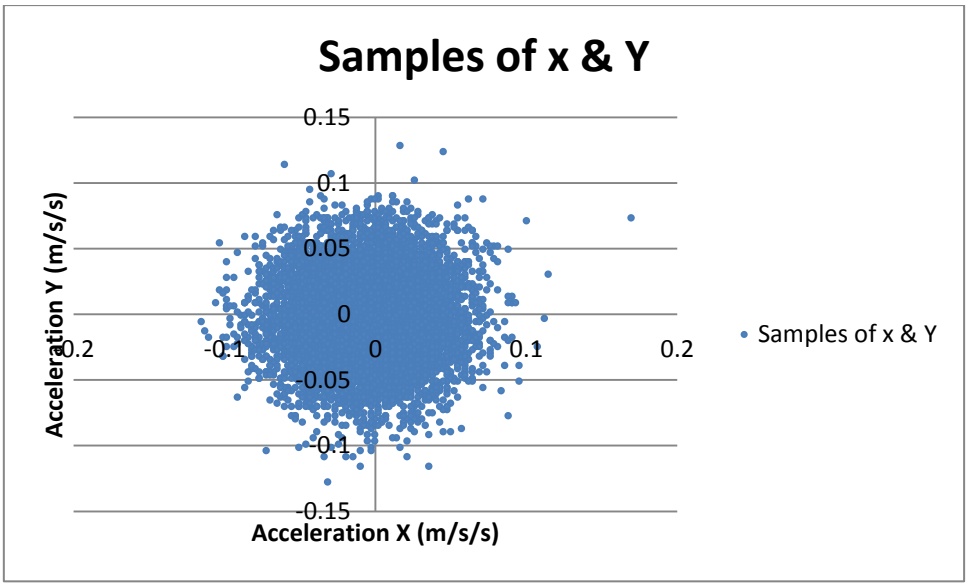

Figure 5-15, Experiment 1 - X & Y Samples

The next part involves converting the acceleration readings to velocity readings. At this point, it was thought and expected that, as roughly all the data points spread more or less evenly above and below the y axis, the velocity would amount to 0 m/s. The acceleration readings are also very small so if the prototype was to convert the acceleration samples to velocity, then it is thought that it would be traveling at a very small speed.

Although when the acceleration values were multiplied by the sample time to convert to velocity, the data suggests that the prototype was moving, even though it was sitting on a bench top stationary. This can be seen in Figure 5-16, Experiment 1 - Velocity X axis and Figure 5-17, Experiment 1 - Velocity Y axis. Both figures indicate that after 8000 samples the prototype was

moving. The x axis being the worst, it thinks the prototype is moving at a velocity of just over -1 m/s by the end of the experiment.

Although, -1 ms^{-1} does not sound like a lot, it was far greater than what was expected or hoped for. It was presumed that the velocity would be in the hundredths. Even the Y axis of the IMU suggesting that it was moving roughly at 0.35 m/s which was greater than what was hoped for.

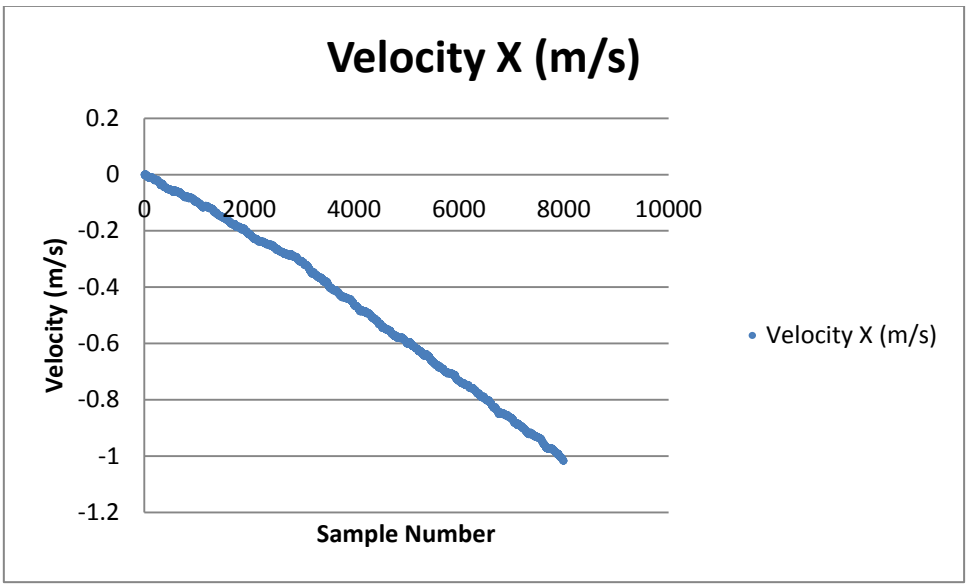

Figure 5-16, Experiment 1 - Velocity X axis

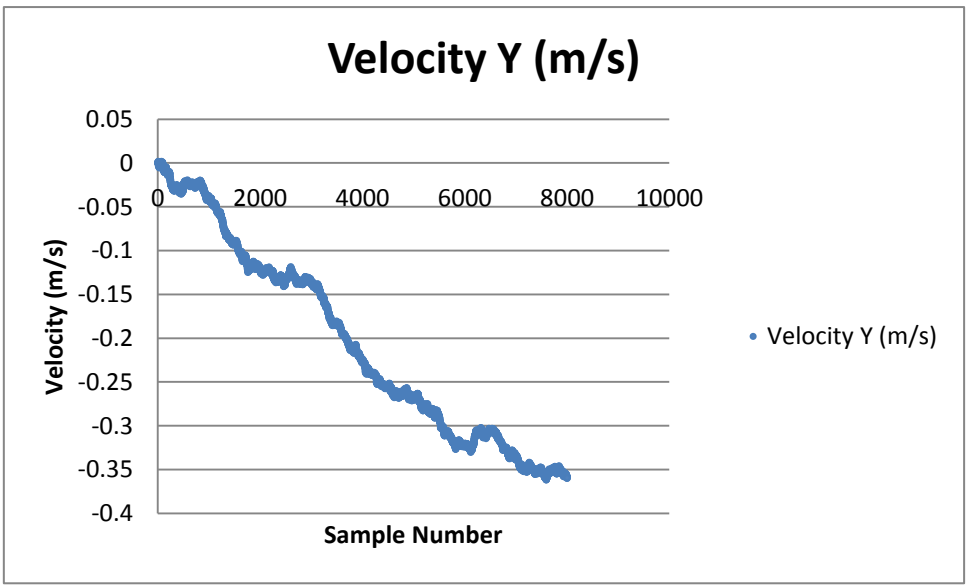

Figure 5-17, Experiment 1 - Velocity Y axis

The X and Y values together can be seen on graph on Figure 5-18, Experiment 1 – Velocity.

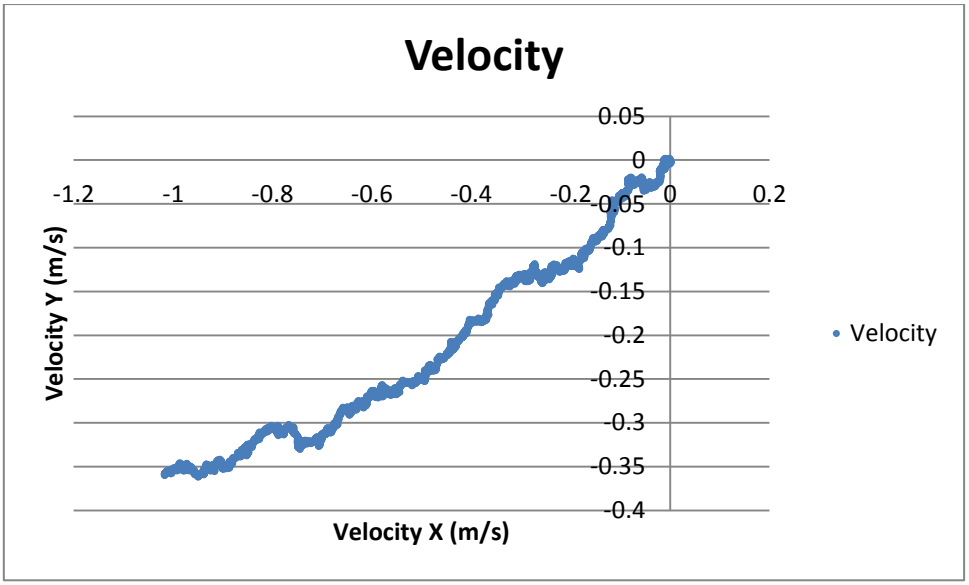

Figure 5-18, Experiment 1 – Velocity

It seems that when the velocity is used to calculate distance travelled, it only seems to make things worse. When looking at Figure 5-19, Experiment 1 - Distance travelled X direction and Figure 5-20, Experiment 1 - Distance travelled Y direction, the graphs represent exponential decay. For the prototype to be simply stationary, the data would strongly suggest that the prototype travelled a long distance.

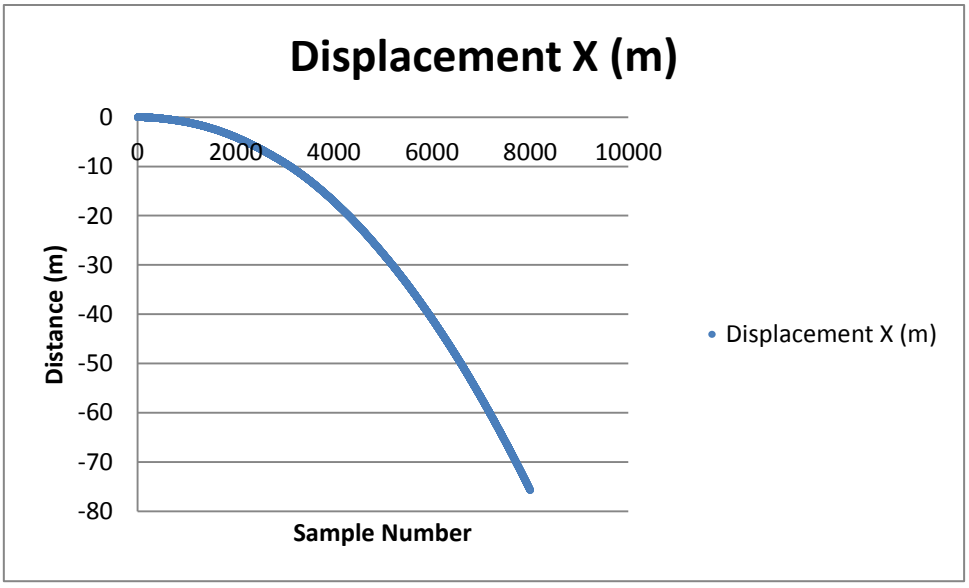

Figure 5-19, Experiment 1 - Distance travelled X direction

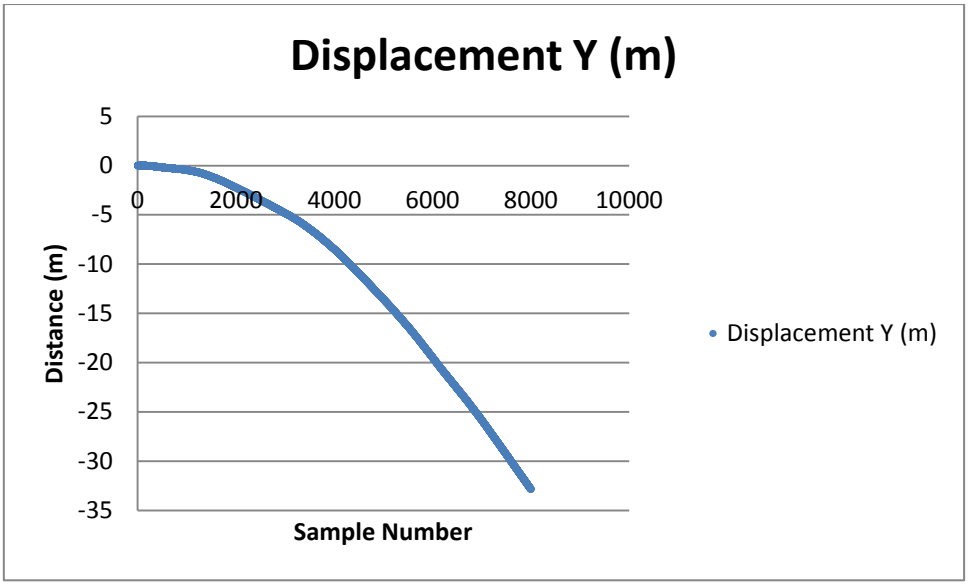

Figure 5-20, Experiment 1 - Distance travelled Y direction

It seems that as the average of the X values was greater than the average Y values it may have caused the greater error in the velocity and then the distance calculations. So if what was thought to be the average value to determine where the 0 point is on the X and Y axis is slightly out, then it will make the distance calculation terribly inaccurate.

The position of the prototype calculated from the IMU's acceleration measurement of the X and Y axis is shown below in Figure 5-21, Experiment 1 - Distance travelled. It claims the prototype has moved far away from its initial position, nearly -80m on the X axis and -35 m on the Y axis form the starting point.

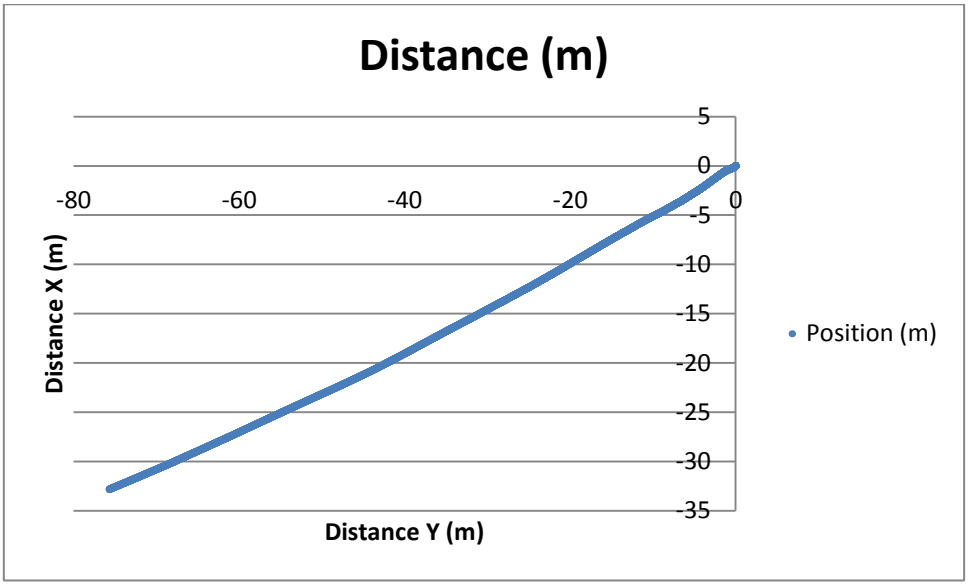

Figure 5-21, Experiment 1 - Distance travelled

5.5.2 Experiment 2

After analysing the data from experiment one above, it was thought that experiment 2 would be pointless. Nevertheless, the data was collected and it was examined. However, instead of completing the experiment 2 only once, it was carried out on three separate occasions.

The velocities of all three experiments are shown below in Figure 5-22, Experiment 2 –Attempt 1 - Velocity, Figure 5-23, Experiment 2 - Attempt 2 - Velocity and Figure 5-24, Experiment 2 - Attempt 3 – Velocity. What the graphs should look like is velocity starting at 0 m/s in both the y and x axis. It should then speed up mostly in the Y axis to a terminal velocity quickly and then hold that velocity for a period of time. Just before the end of the graph, the velocity should decline until it reaches 0 m/s on the Y axis. There should be minimal movement in the X axis as this experiment only involves moving in a straight line.

The results do not show what was expected. The graph shows the object speeding up as expected but the all three graphs show a linear increase in speed where it should depict a straight horizontal line. All three graphs show a deceleration at the end, which are close to the heights of the accelerations at the beginning. This part does look good; however, as they are all a starting at a point much hirer, when the prototype does come to a complete stop, the data suggests that it is still moving.

It appears that the likely cause of this error is that the calibration is slightly out. Therefore, when the prototype is sitting still and is not moving, the small calibration error would make it think it is moving. Another reason could be that the ground is not perfectly flat and therefore allows gravity to affect the X and Y axes. The calibration method used did not correct the acceleration readings properly and subsequently the acceleration readings are slightly wrong.

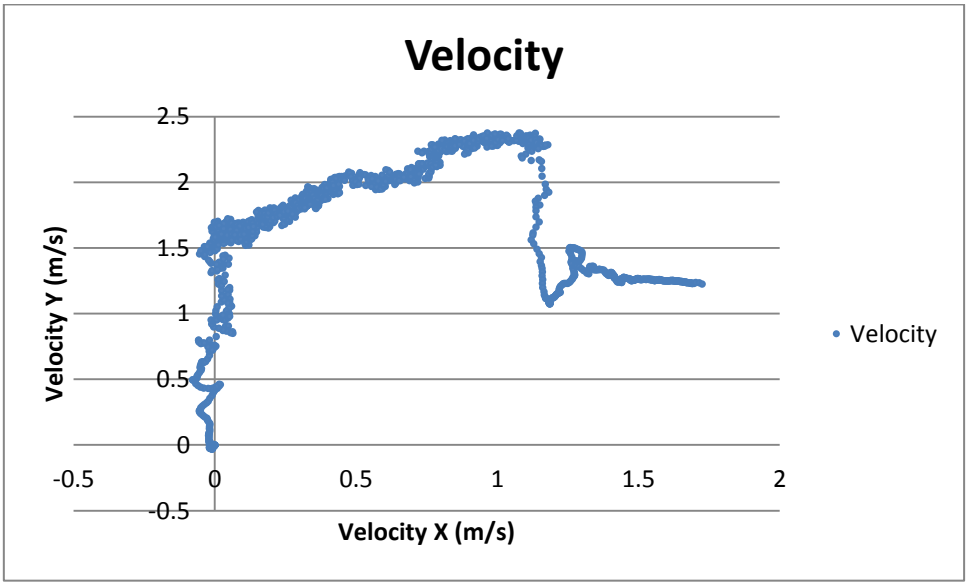

Figure 5-22, Experiment 2 –Attempt 1 - Velocity

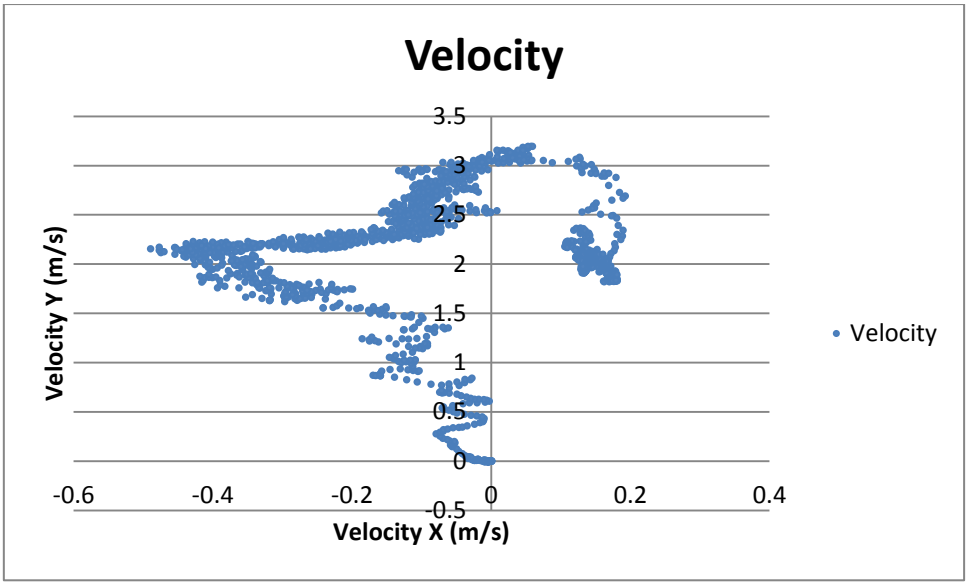

Figure 5-23, Experiment 2 - Attempt 2 - Velocity

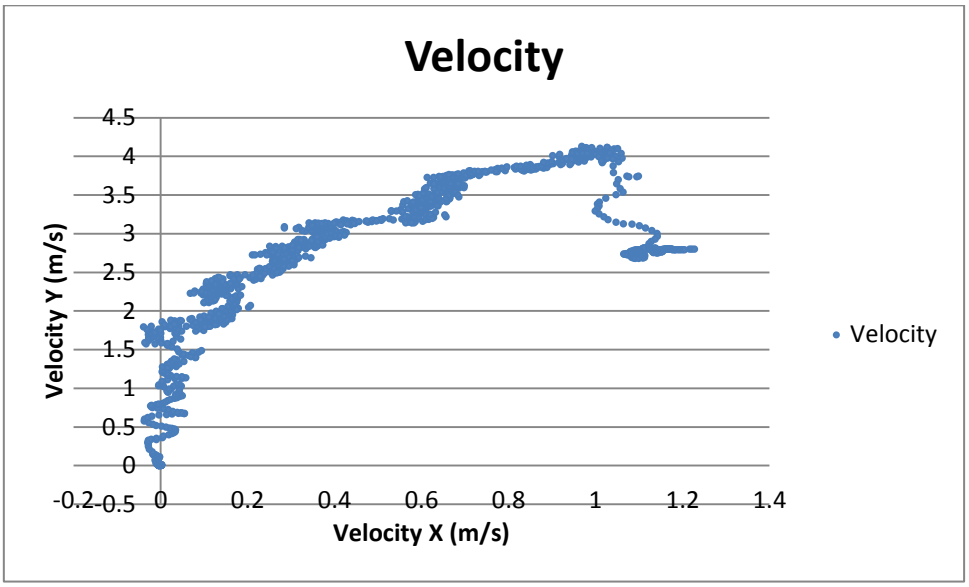

Figure 5-24, Experiment 2 - Attempt 3 – Velocity

As the velocity graphs are very wrong, the distance graphs are also wrong. This is seen in Figure 5-25, Experiment 2 - Attempt 1 – Position, Figure 5-26, Experiment 2- Attempt 2 – Position, Figure 5-27, Experiment 2 - Attempt 3 - Position. The distance covered should have been 28.126 meters. The results clearly show that the prototype has significantly over shot the distance it should have. The distances the prototype thinks it has travelled covered in attempt 2 and 3 is double what it actually did.

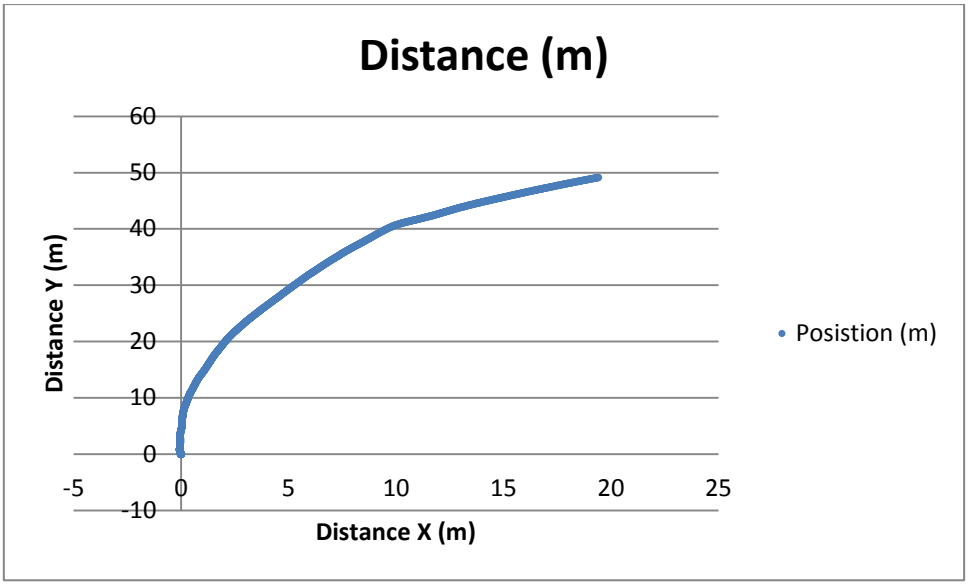

Figure 5-25, Experiment 2 - Attempt 1 – Position

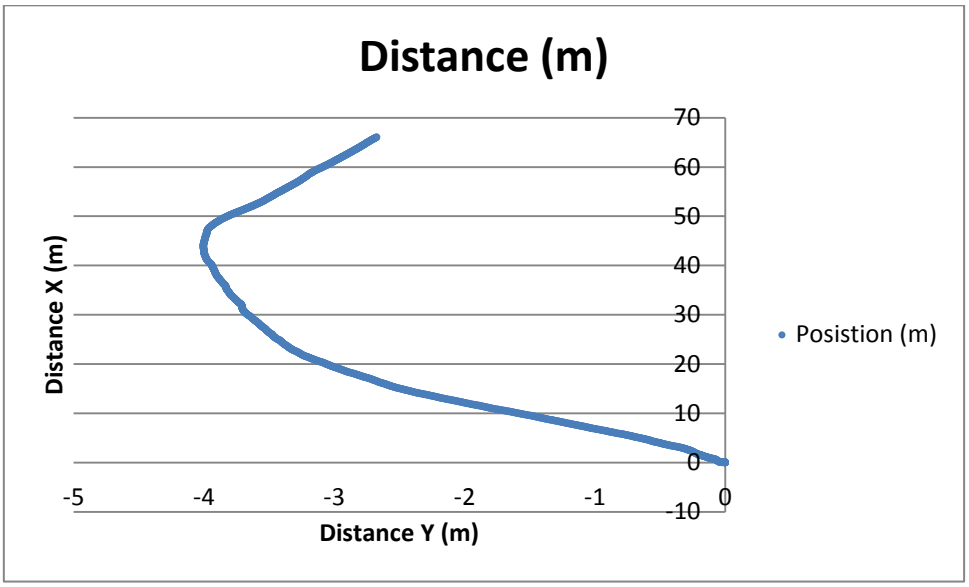

Figure 5-26, Experiment 2- Attempt 2 – Position

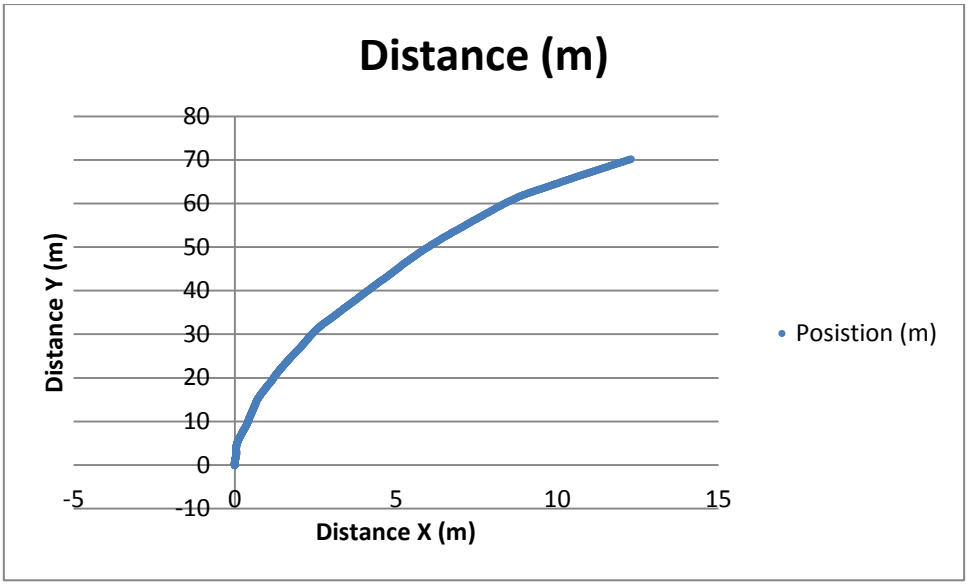

Figure 5-27, Experiment 2 - Attempt 3 - Position

5.6 Correction to Results

There was an error in the sketch used in the previous experiment. The variable used to calibrate the data was an integer. This meant that the 32 bit calibration value could never have decimal places. This would result in the calibration being very slightly out as it would round to the nearest whole number. Which meant that all the data collected would need to be adjusted slightly as the 32 bit value in mg-force would require a decimal value to be precise.

In experiment 1, where the prototype was stationary on the bench, the calibration value was then adjusted by the average acceleration value. The X average (shown in Table 5-2, Experiment 1 - X axis) was used to adjust the calibration value on the X axis. The Y average (shown in Table 5-3, Experiment 1 - Y axis) was used to adjust the calibration value on the Y axis. The value of calibrating the x axis was altered from -1571 to -1571.006352. The value for calibrating the y axis was changed from 169 to 168.997753.

The velocity in the X axis and Y axis are shown below in Figure 5-28, Experiment 1 - X axis and Figure 5-29, Experiment 1 - Y axis. Using slightly adjusted velocity values has made a significant improvement in the velocity values. The values fluctuate around the 0 m/s mark as expected.

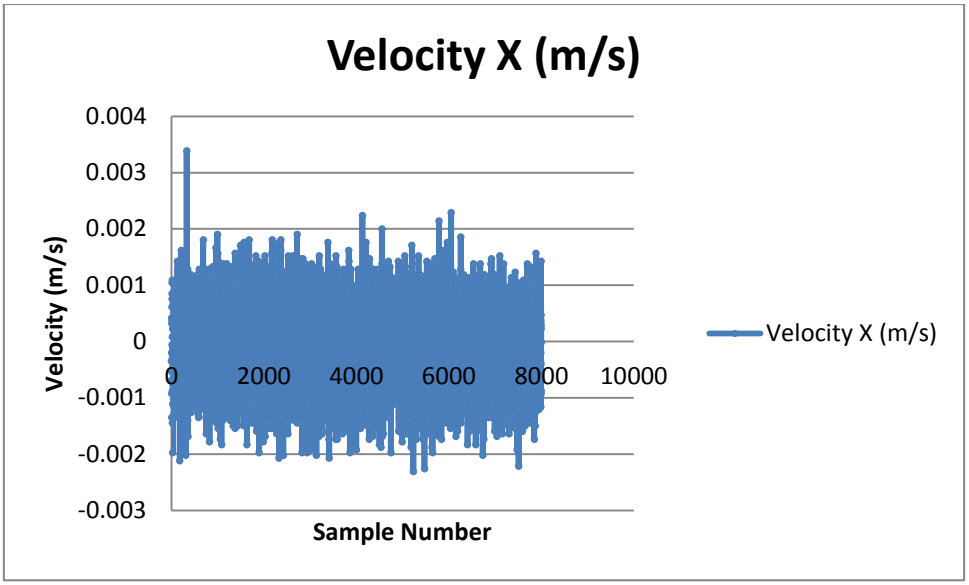

Figure 5-28, Experiment 1 - X axis

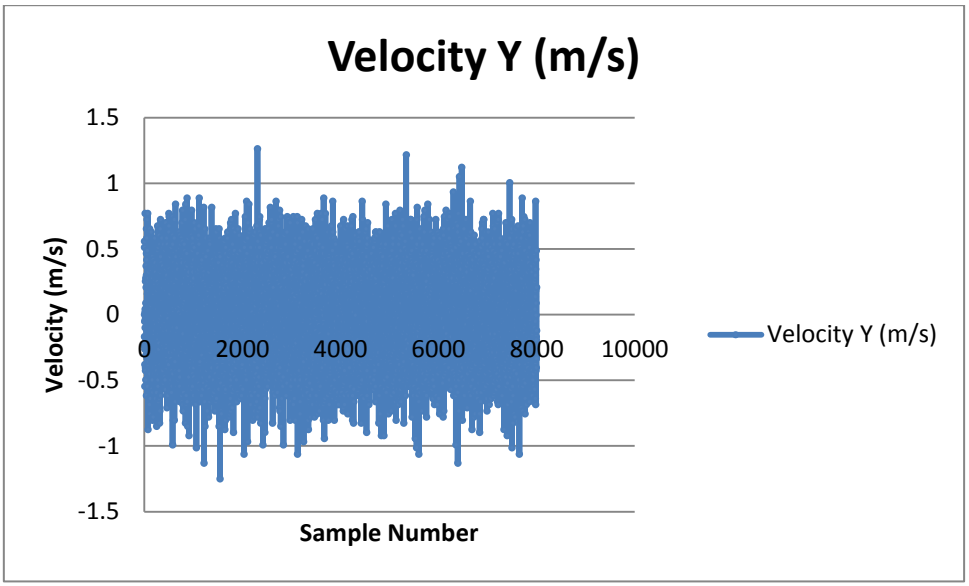

Figure 5-29, Experiment 1 - Y axis

The distance was also dramatically improved. As can be seen in Figure 5-30, Experiment 1 – X axis and Figure 5-31, Experiment 1 – Y axis, the distance on both axis amount to 0 meters.

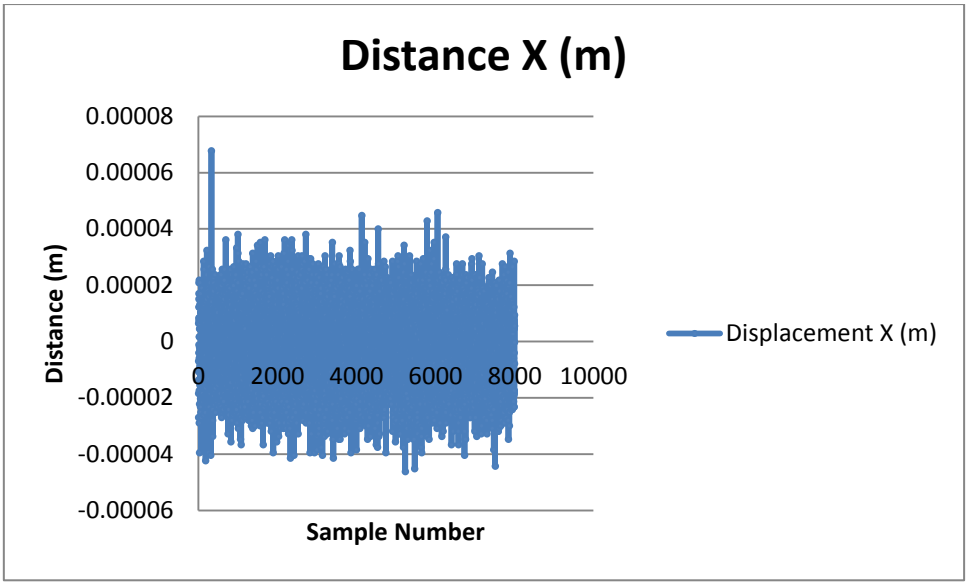

Figure 5-30, Experiment 1 – X axis

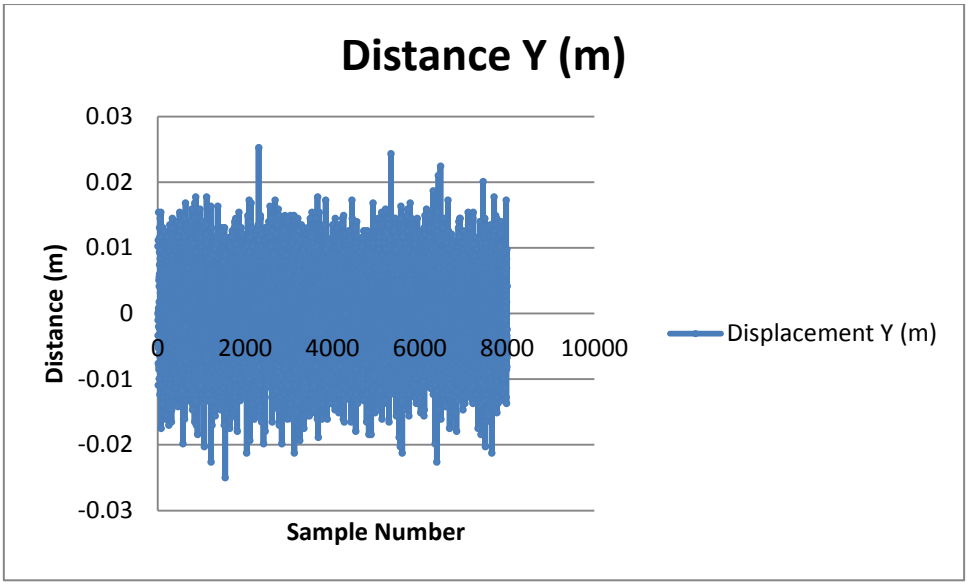

Figure 5-31, Experiment 1 – Y axis

The result of the calibration value being a decimal number made the prototype significantly more accurate. This resulted in the sketch begin changed so that the calibration value was a single precision floating point number instead on an integer. The calibration values could go to 4 decimal places. This meant that the calibration should be much more precise than before and therefore minimise the error.

Experiment 1 was conducted again. The prototype was stationary on the bench and data was collected. The calibration value for the X axis was -1429.2430 and for the Y axis 16.711.

The results of the samples are shown below in Figure 5-32, Experiment 1 - X axis and Figure 5-33, Experiment 1 - Y axis. Again the results appear to be good as samples are around the 0 m/s/s for both axes.

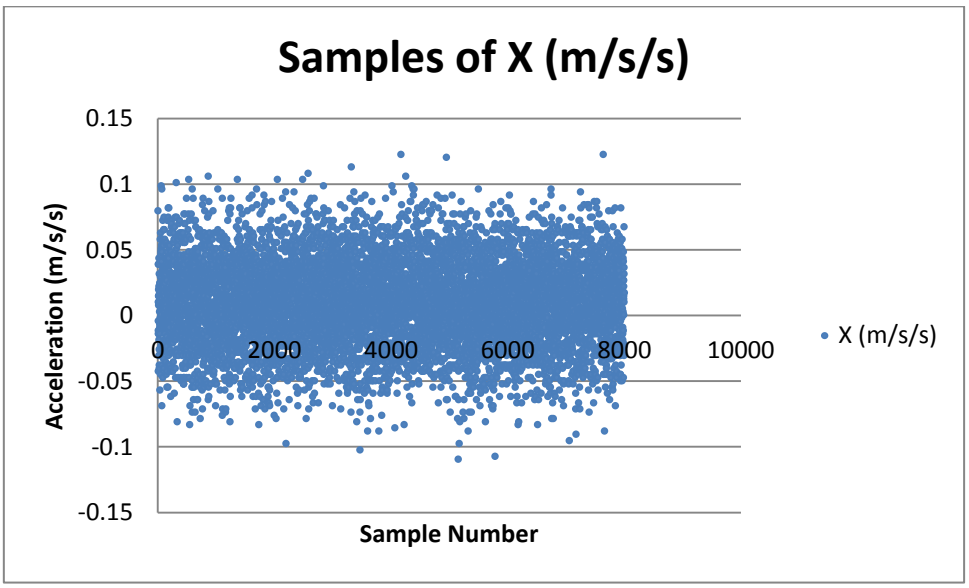

Figure 5-32, Experiment 1 - X axis

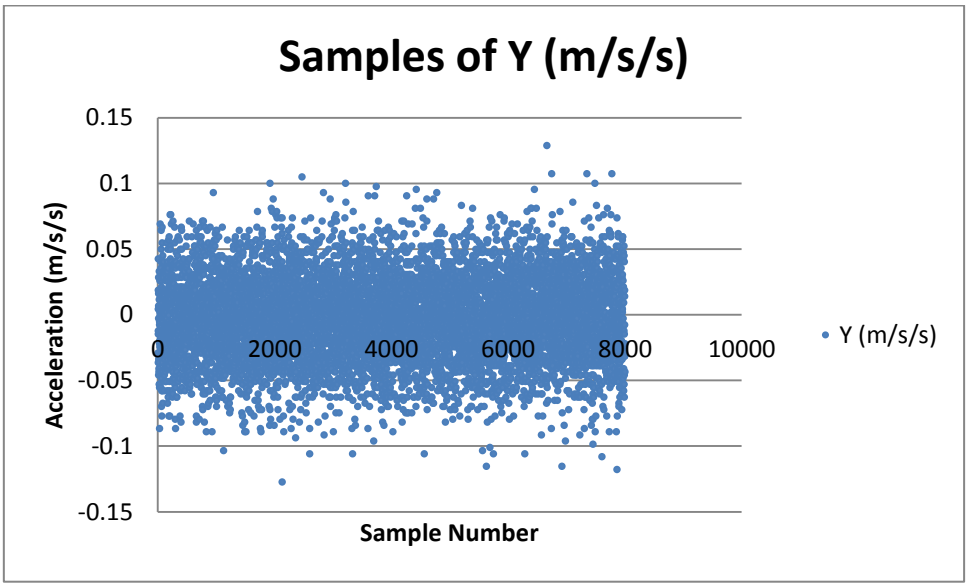

Figure 5-33, Experiment 1 - Y axis

Important aspects of the sample data are shown in Table 5-4, Experiment 1 - X axis and Table 5-5, Experiment 1, Y axis. What stands out is the average X and Y is bigger than expected. The values were expected to be smaller because the addition of the calibration value to 4 decimal values. It appears not to have affected the results.

Total Samples of X	
Maximum X value:	0.122848283
Minimum X value:	-0.10938928
Average X value:	0.00959548
Standard Deviation:	0.031990765

Table 5-4, Experiment 1 - X axis

Total Samples of Y	
Maximum Y value:	0.128867246
Minimum Y value:	-0.12731233
Average Y value:	-0.00303896
Standard Deviation:	0.031354717

Table 5-5, Experiment 1, Y axis

The results appear to be no better than before in experiment 1. This can be seen in Figure 5-34, Experiment 1, X axis and Figure 5-35, Experiment 1, Y axis. In both axes, the prototype thinks its speeding up when it is just stationary. The addition of four decimal places did nothing to improve the results.

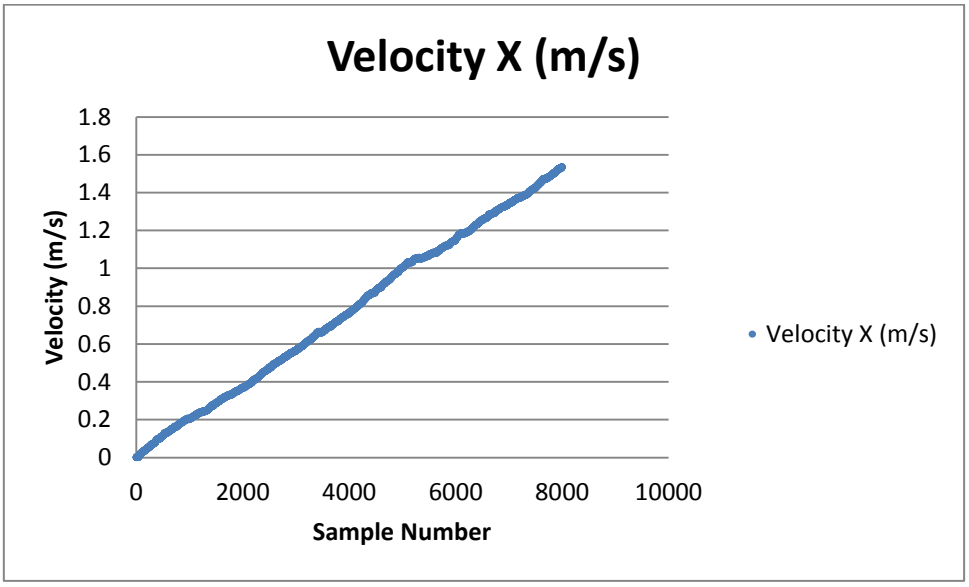

Figure 5-34, Experiment 1, X axis

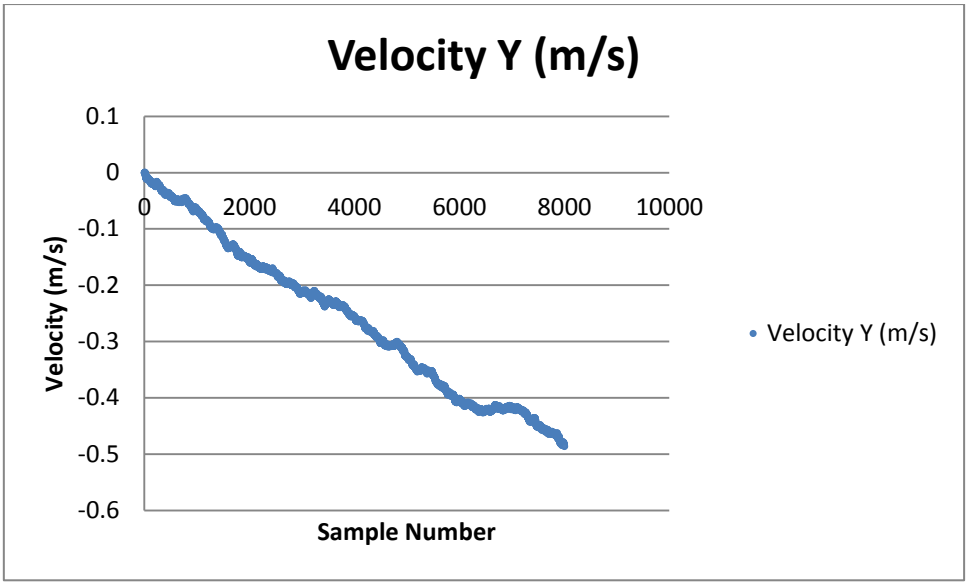

Figure 5-35, Experiment 1, Y axis

As expected, the results are that distance travelled in both the X and Y axes is significantly out. The X axis is to the most incorrect suggesting that it travelled 120 meters in the space of approximately 2 minutes and 40 seconds. The distance travelled in the X direction is shown below in Figure 5-36, Experiment 1 - X axis and distance travelled in the Y direction is shown below in Figure 5-37, Experiment 1 - Y axis.

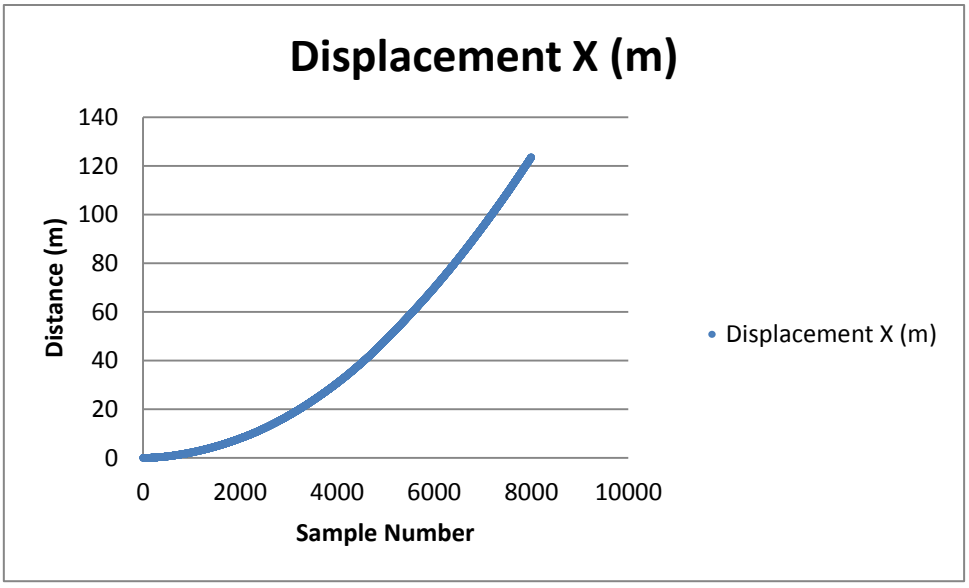

Figure 5-36, Experiment 1 - X axis

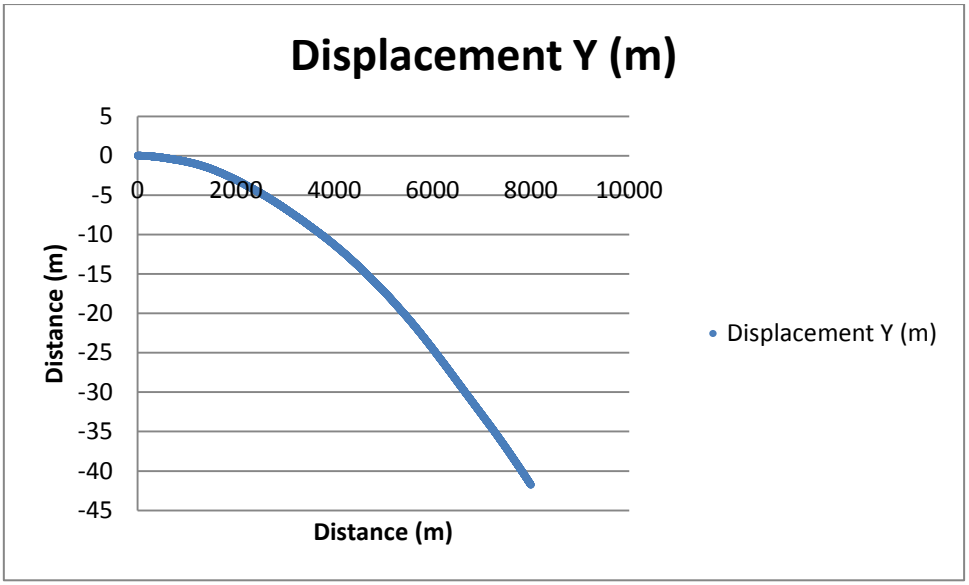

Figure 5-37, Experiment 1 - Y axis

6 Conclusion

The small error in the calibration makes the prototype extremely inaccurate. If the calibration is out by the smallest of margins then every calculation keeps adding the error on top of error. As 8000 samples were taken, there were 8000 small errors added on to each calculation. The double integration of acceleration used to calculate distance only makes the error worse. As in experiment 1 with 4 decimal points added to the calibration, the prototype thought it had moved 120 m on the X axis and over -40 m on the Y axis when it was in a stationary position.

It may be that collecting 250 samples for calibrating may not be enough. Perhaps more is need to get an adequate calibration value. The more taken, the more likely a precise value could be obtained for calibration. As the calibration is only marginally out, this could be enough to make the prototype accurate.

Another way of making the prototype more accurate would be to add more decimal places to the calibration value. The more decimal places that it is able to have, then the more accurate the calibration will become. The only problem with adding more decimal places with the current system is the Arduino Due may not be the best microcontroller for the job. That is because a variable has to be a float to be able to have decimal places. The Arduino is only accurate to 6 to 7 digits (not bits) [39]. Even if the variable is changed to double making it a bigger 64 bit number instead of a 32 bit. However, it will not gain in precision [40].

A different microcontroller may be needed that can calculate floating variables with more precision. The controller would need to go beyond 6 to 7 values of precision to be a better than the Arduino Due used in this project. Therefore, a 10 or 12 value precision would be better and could possible produce improved results.

The Z axis acceleration readings from the IMU were not used or recorded throughout the experiments of this project. This should have been used and may have affected the results if vectors were used to calculate distance. The reason it was left out was because there was only a limited

amount of storage space on the EEPROM and was thought that the Z axis was not that important as the prototype would not be flying or moving vertically. However, it was not used and therefore is not known if it would or would not make the results more accurate.

Lastly, if more memory was available, then the IMU acceleration readings could be samples at its maximum rate of 1 kHz. Then the data interrupt on the IMU could be used to signal the microcontroller that new data is available every time it has been updated. The wire from the IMU interrupt pin is already connected to the Arduino Due so there is no need to do any more wiring in that regard.

The only issue is having a big enough storage space to store the samples. Currently, it is recording values at 50 Hz and takes about 2 minutes and 40 seconds before the 32 kilobyte memory is full. Therefore, a much bigger storage space is needed. It would be suggested to use around 500 megabytes. That is because if 2 bytes for each axis used at a rate of 1 kHz, at one second it would have stored 6 000 bytes. In ten minutes, it would have stored 360 000 bytes. Therefore, if the acceleration samples were saved to the EEPROM at their quickest rate, then much more storage space will be needed.

It is unlikely that it would affect the experiment 1 in a positive way. As the device is not moving then the sample rate is not critical. It could possibly make the results more accurate for the Experiment 2 and other future experiments to be carried out. In particular, if the prototype is later used for calculating distance for a faster accelerating object.

7 Future Recommendations

The dead reckoning project did not work anywhere near as well as expect and problems were caused by inaccurate calibration. It is recommended that if that this project was to be carried out again that much focus be on this part. The straight forward test of leaving the accelerometer on the bench and double integrating the acceleration readings to calculate distance needs to be worked on first before headway can be made in moving experiments. It is recommended that more samples are taken to calibrate the sensor each time an experiment is conducted. Currently, only 250 samples were taken and therefore experiments should be conducted with taking 500, 750 then 1 000 samples to see if it makes a difference.

Following collecting more data for the calibrating the accelerometer, it is recommended to incorporate the Z axis of the accelerometer into the readings as well as the magnetometer. The Z axis will then be able to be used to produce vector quantities and the magnetometer will be used for working out direction. For instance, it could be used for working out if the prototype is heading in the north or south direction. It then could be plotted on a graph.

If calculating the calibration value worked by using more samples does not work, a different controller may need to be sought. At the moment the Arduino Due can only calculate floating values to 6 to 7 values precision. As the calibration values were only marginally out, this could be difference for the prototype working or not. Therefore, a controller with more precision with floating points could produce better results. The decimal values could go way beyond 4 decimal places and make the calibration extremely accurate. Therefore, if the calibration has less error then the distance calculation would much better.

There could be much more memory added to the prototype. For conducting longer and possibly more accurate experiments more data would be needed. This will give the user the ability to load more raw data into Excel to be analysed. It would be able to be used to unlock the full potential of the prototype which left out the Z axis of the accelerometer and the magnetometer.

Finally, the sketches created for this prototype could be written in a better way so that if altering certain parts of the program was needed - this could be done so without needing to rewrite lots of code. Functions were created in some areas but in hindsight, more could have been used. The code is also in two separate parts, the sampling data part and the uploading to Excel part. It would be ideal to have both of these parts incorporated into one sketch to save time of always having to upload the different sketches to the Arduino Due.

The LCD was only working right at the end of the project. The displays were therefore put together very quickly and could have been used in a more intuitively way by the operator. This would be a fairly straight forward task as the code used to write to this device is not complicated.

8 Bill of Parts

Part	Cost	Qty	Total	Supplier
IMU – MPU 9150	34.95	1	$34.95	Little Bird Electronics
Accelerometer ADXL345	$17.95	1	$17.95	Little Bird Electronics
Logic Level Converter	$2.95	1	$2.95	Little Bird Electronics
I2C/TWI LCD1602 Module	$19.36	1	$19.36	Little Bird Electronics
I2C EEPROM - 256kbit	$2.09	1	$2.09	Little Bird Electronics
USB to Micro USB	$5.95	1	$5.95	Jaycar
ADAP PLG 2.1mm DC	$4.95	1	$4.95	Jaycar
Batt ALK 9V Panasonic	$5.95	1	$5.95	Jaycar
Jumper set PLG/PLG	$3.95	2	$7.90	Jaycar
Momentary Button	-	1	-	Supplied By Murdoch
Arduino Due	$56.99	1	$54.95	Supplied By Murdoch
Headers	-	Set	-	Supplied By Murdoch
Circuit Board	-	1	-	Supplied By Murdoch
Wires	-	-	-	Supplied By Murdoch

Table 8-1, Bill of Parts

9 Appendix A: Software

The sketch was produced in IDE 1.5.7 using Arduino Due. The prototype had two separate sketches. One for collecting samples of acceleration and the other to extract the data to be used in an excel spread sheet. The sampling sketch is shown first below.

9.1 Collecting Samples

/ This sketch was based on previous sketches and libraries made by other people.

// The final product was done by James Baker for a Dead Reckoning Project at Murdoch University.

//==
============================

// I2C device class (I2Cdev) demonstration Arduino sketch for MPU9150

// 1/4/2013 original by Jeff Rowberg <jeff@rowberg.net> at https://github.com/jrowberg/i2cdevlib

// modified by Aaron Weiss <aaron@sparkfun.com>

//

// Changelog:

// 2011-10-07 - initial release

// 2013-1-4 - added raw magnetometer output

/* ==

I2Cdev device library code is placed under the MIT license

Permission is hereby granted, free of charge, to any person obtaining a copy

of this software and associated documentation files (the "Software"), to deal

in the Software without restriction, including without limitation the rights

to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

copies of the Software, and to permit persons to whom the Software is

furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in

all copies or substantial portions of the Software.

```
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
===============================================
*/
//************************************************************************************
******************
// header files to be included
#include "Wire.h"
#include "I2Cdev.h"
#include "MPU6050.h"
#include "LiquidCrystal_I2C.h"



//************************************************************************************
******************
// the address of the IMU 0x68 and memory
MPU6050 accelgyro;  //need help here

volatile uint8_t Mem_address_I2C = 0x50;      // address of memory on the I2C bus (50 in hex)
volatile int address_location_mem = 0;       // address where data is to be sent on memory (it has
upto 32k)

volatile uint8_t IMU_address_12C = 0x68;      // address of IMU on the I2C bus (68 in hex)
```

```
uint8_t register_56_Interrupt_Enable = 0x38;   //address of register to enable interrupt when new data is available

uint8_t bit_number = 0;                 //location to store bit on register

uint8_t Interrupt_on = 1;               //turn on interrupt

bool state;                         //state true or false if bit was successfully changed

int16_t ax, ay, az;


//*************************************************************************************************

// momentary button configuration

int momentary_button = 3;

bool toggle_condition = LOW;

int buttonState;

int lastButtonState = LOW;

int reading = 0;

long lastDebounceTime = 0;

long debounceDelay = 30;

bool wait_for_button = true;

bool only_once_please = true;

bool only_once = true;



bool memory_full_once = true;

bool finished_collecting = false;

int choose_case;


int sampleNo = 1;

bool data_collected = LOW;
```

```
bool begin_recording_data = false;

unsigned long last_time = 0;

unsigned long start_time = 0;

//*********************************************************************************
******************

LiquidCrystal_I2C lcd(0x20,16,2);

int once_LCD_wait = 1;

int begin_LCD = 1;

int count_down_LCD = 5;

unsigned long LCD_Display_XY = 0;

int LCD_Finished = 1;

unsigned long LCD_count_down_time = 0;

unsigned long average_XY = 0;

float ax_average = 0;

int ax_average_non_dec = 0;

int ax_average_dec = 0;

float ay_average = 0;

int ay_average_non_dec = 0;

int ay_average_dec = 0;

float divide_xy_for_average = 0;

float ax_total = 0;

float ay_total = 0;

//*********************************************************************************
******************

void setup(){

  Wire.begin();     //initialize I2C bus

  lcd.init();

  lcd.backlight();
```

```
 Serial.begin(9600);      //initialize serial communication

 accelgyro.initialize();

 Serial.println("Initializing I2C devices...");     // initialize device

 Serial.println("Testing device connections...");    // verify connection

 Serial.println(accelgyro.testConnection() ? "MPU9150 connection successful" : "MPU9150
connection failed");

 pinMode(momentary_button, INPUT);

}

//**************************************************************************************
******************

void loop(){

 if (once_LCD_wait == 1){lcd.clear(); lcd.print("Ready..."); once_LCD_wait = 0;}

 if (toggle_condition == HIGH){ choose_case = 1;} else{ choose_case = 2;}

 if (wait_for_button == true){

 int reading = digitalRead(momentary_button);

 Debounce_button_Software (reading);

 lastButtonState = reading;

 }


 switch (choose_case){


 case 1:

if (begin_LCD == 1){LCD_Finished = 1;

 if ((millis() - average_XY ) >= 20)

    { average_XY = millis(); accelgyro.getAcceleration(&ax, &ay, &az); divide_xy_for_average =
divide_xy_for_average + 1;

    ax_total = ax_total + ax; ay_total = ay_total + ay;

    //Serial.print(ax_total); Serial.print("\t"); Serial.print(ay_total); Serial.print("\t");
```

```
    //ax_average = (ax_total)/divide_xy_for_average; ay_average =
(ay_total)/divide_xy_for_average;

    //Serial.print(ax_average); Serial.print("\t"); Serial.print(ay_average); Serial.print("\t");
Serial.println(divide_xy_for_average);

    //ax_average_non_dec = ax_average; ay_average_non_dec = ay_average;

    //ax_average_dec = (10000*(ax_average - ax_average_non_dec)); ay_average_dec =
(10000*(ay_average - ay_average_non_dec));



}

if ((millis() - LCD_count_down_time) >= 1000){

LCD_count_down_time = millis();

lcd.clear(); lcd.print("begin in "); lcd.print(count_down_LCD); count_down_LCD = count_down_LCD -
1; if (count_down_LCD == -1){begin_LCD = 0;

ax_average = (ax_total)/divide_xy_for_average; ay_average = (ay_total)/divide_xy_for_average;

//Serial.println(ax_total); Serial.println(divide_xy_for_average); Serial.println(ay_total);
Serial.println(divide_xy_for_average);

//Serial.println(ax_average); Serial.println(ay_average);

ax_average_non_dec = ax_average; ay_average_non_dec = ay_average;

ax_average_dec = (10000*(ax_average - ax_average_non_dec)); ay_average_dec =
(10000*(ay_average - ay_average_non_dec));

Serial.println(ax_average_non_dec);Serial.println(ax_average_dec);
Serial.println(ay_average_non_dec); Serial.println(ay_average_dec);

delay(5); Write_data(ax_average_non_dec); delay(5); Write_data(ax_average_dec); delay(5);
Write_data(ay_average_non_dec); delay(5); Write_data(ay_average_dec);}}

}



else {

 if ((millis() - last_time) >= 20){

   //Serial.println((millis() - last_time));

   last_time = millis();
```

```
 if (sampleNo < 8000){
  accelgyro.getAcceleration(&ax, &ay, &az); delay(5);
  //Serial.print("Sample x/y:"); Serial.print("\t"); Serial.print(sampleNo); Serial.print("\t");
Serial.print(ax); Serial.print("\t"); Serial.println(ay); delay(5);
  Write_data(ax); delay(5); Write_data(ay); delay(5);
  sampleNo = sampleNo + 1;
  memory_full_once = true;
  finished_collecting = true;
  if ((millis() - LCD_Display_XY) >= 500){
  lcd.clear(); lcd.print(ax); lcd.setCursor(0, 1); lcd.print(ay);
  LCD_Display_XY = millis();
  }
 }
 else {
  if (memory_full_once == true){
  //Serial.print("memory is full");
  memory_full_once = false;
  lcd.clear(); lcd.print("Memory is full");
  }
 }
 }
}
 break;

 case 2:
 if (finished_collecting == true){
 if (LCD_Finished == 1){lcd.clear(); lcd.print("Finished."); lcd.setCursor(0, 1); lcd.print("Begin
again..."); LCD_Finished = 0;
```

```
 begin_LCD = 1; count_down_LCD = 5; divide_xy_for_average = 0; ax_total = 0; ay_total = 0;
ax_average = 0;  ay_average = 0;}

 //Serial.print("Finished collecting data");

 if (sampleNo < 7998){

 Write_data(1); delay(5);

 Write_data(1); delay(5);

 Write_data(1); delay(5);

 Write_data(1); delay(5);

 Write_data(1); delay(5);

 Write_data(1); delay(5);

 sampleNo = sampleNo + 3;

 }

 finished_collecting = false;

 }

 break;

}


}

void Debounce_button_Software (int reading)

{

if (reading != lastButtonState)

 {

   lastDebounceTime = millis();

  }

      if ((millis() - lastDebounceTime) > debounceDelay)

        {

         if(reading != buttonState)

           {
```

```
            buttonState = reading;
            if (buttonState == HIGH)
             {
              toggle_condition = !toggle_condition;
              Serial.println("button pressed");
              }
            }
          }
}
void Write_data (int data){
    Wire.beginTransmission(Mem_address_I2C);              //begins transmission with eeprom and requires its 7 bit address
    Wire.write(highByte(address_location_mem));           //sends address of high byte first of where data will be stored on the eeprom
    Wire.write(lowByte(address_location_mem));      //sends address of low byte second of where data will be stored on the eeprom
    Wire.write(lowByte (data));
    Wire.write(lowByte (data >> 8));
    Wire.endTransmission();                         //stops transmission
    address_location_mem = address_location_mem + 2;

}
```

9.2 Extracting Data

The following sketch was used to extract data form the EEPROM:

```
#include <Wire.h>

#include "LiquidCrystal_I2C.h"


//**************************************************************************************************************************

uint8_t Mem_address_I2C = 0x50;

int address_location_mem = 0;

boolean memory_full = true;


const int button = 3;


int toggle_condition;

int buttonState;

int lastButtonState = LOW;

int reading = 0;

int SampleNo = 0;

long lastDebounceTime = 0;

long debounceDelay = 50;


int16_t total;

byte lowvalue;

byte highvalue;

uint8_t two_bytes = 2;


LiquidCrystal_I2C lcd(0x20,16,2);
```

```
unsigned long LCD_Dot = 0;

int dot = 0;

//*********************************************************************************
*********************************************************************************
**********************


void setup()

 {

   Wire.begin();

   Serial.begin(9600);

   lcd.init();

   lcd.backlight();

   pinMode(button, INPUT);

   lcd.clear(); lcd.print("Ready to"); lcd.setCursor(0, 1); lcd.print("Download");

  }



void loop()

   {

    int reading = digitalRead(button);

    Debounce_button_Software (reading);

    switch(toggle_condition)

         {

           case 0:

           //Serial.println("do nothing");

           break;

           case 1:

           if ( SampleNo <= 7999 )
```

```
{
    if (SampleNo < 3){
    Serial.print("Average x: "); Serial.print("\t"); Serial.print(",");  delay(5);
    read_value(); delay(5); Serial.print(total); Serial.print(",");
    read_value(); delay(5); Serial.println(total);
    SampleNo = SampleNo + 1;
    Serial.print("Average y: "); Serial.print("\t"); Serial.print(",");  delay(5);
    read_value(); delay(5); Serial.print(total); Serial.print(",");
    read_value(); delay(5); Serial.println(total);
    SampleNo = SampleNo + 1;
    }
    else {
    Serial.print("Sample x/y: "); Serial.print("\t"); Serial.print(SampleNo); Serial.print("\t");
Serial.print(",");
    delay(5);
    read_value();
    delay(5);
    Serial.print(total); Serial.print(",");
    delay(5);
    read_value();
    delay(5);
    Serial.println(total);
    SampleNo = SampleNo + 1;
    if ((millis() - LCD_Dot) >= 1000){
    LCD_Dot = millis();
    dot = dot + 1;
      if (dot == 1){lcd.clear(); lcd.print("Downloading.");}
      if (dot == 2){lcd.clear(); lcd.print("Downloading..");}
```

```
            if (dot == 3){lcd.clear(); lcd.print("Downloading...");}
            if (dot == 3){dot = 0;}
           }
          }
        }
         else if (memory_full == true && SampleNo == 8000 )
         {
          Serial.println();
          Serial.print("memoryfull");
          memory_full = false;

         }
        else
         {
          //empty. Once the memory is full the arduino will stay in continuous loop of doing nothing
         }
        break;
        default:
        {
        }
       }
    lastButtonState = reading;
    delay(5);
  }
void Debounce_button_Software (int reading)
{
if (reading != lastButtonState)
```

```
 {
   lastDebounceTime = millis();
  }
      if ((millis() - lastDebounceTime) > debounceDelay)
       {
        if(reading != buttonState)
          {
           buttonState = reading;
           if (buttonState == HIGH)
             {
              toggle_condition = !toggle_condition;
              Serial.println("button pressed");
              }
          }
        }
}
//*************************************************************************************************************************
// PROJECT:      Designed for Dead reckoning projected
//
// PURPOSE:      Reads the value stored in the 24LC256 memory. Only 8 bits of the 16 bits of the acceleration reading is stored in one location.
//               This will read only the lower 8 bits of the acceleration reading. Additionally the "," are implemented to separate values when putting in excel.
//
// CREATED BY:   James Baker
// DATE:         5/10/2014
//
```

```
//*************************************************************************************
*************************************************************************************


 void read_value ()

{


    Wire.beginTransmission(Mem_address_I2C);              //begins transmission with eeprom and
requires its 7 bit address

    Wire.write(highByte(address_location_mem));           //sends address of high byte first of
where data will be stored on the eeprom

    Wire.write(lowByte(address_location_mem));            //sends address of low byte second of
where data will be stored on the eeprom

    Wire.endTransmission();                          //stops transmission

    Wire.requestFrom(Mem_address_I2C, two_bytes);              //request a byte form the
eeprom from the address mention previiously

    lowvalue = Wire.read();

    highvalue= Wire.read();

    delay(5);

    total = (highvalue << 8) | lowvalue;

    address_location_mem = address_location_mem + 2;


}
```

10 Appendix B: Calculations in Excel

This section describes the formulas used in excel. The calibration value used to correct the acceleration readings due to gravity was done so by the Arduino Due. It sampled acceleration data for 5 seconds at a rate of 50 Hz and added them all together. At the end the value was divided by the number samples taken. This was then used to correct the raw data before calculations were performed.

The step was to convert the 16 bit signed values to as scales of $\pm 2g$. The following calculation was used:

$$g\ force = value \times \frac{4}{65536} \qquad \text{Equation 10-1}$$

Once the g-force was calculated, the acceleration could be calculated in SI units of m/s/s. the equation used was:

$$acceleration = g\ force \times 9.80665 \qquad \text{Equation 10-2}$$

The acceleration was then integrated into velocity using the dt equalling 50 Hz (0.02). The following equation was used to calculated velocity:

$$velocity = acceleration \times dt + previous\ velocity \qquad \text{Equation 10-3}$$

The velocity then could be integrated into distance. The following formula was used to work out distance:

$$distance = velocity \times dt + previous\ distance \qquad \text{Equation 10-4}$$

References

[1] “Global Positioning System,” Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Global_Positioning_System. [Accessed 15 12 2014].

[2] “Microelectromechanical systems,” Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Microelectromechanical_systems. [Accessed 15 12 2014].

[3] “Dead Reckoning,” 12 07 2014. [Online]. Available: http://en.wikipedia.org/wiki/Dead_reckoning. [Accessed 21 7 2014].

[4] “Inertial Navigation System,” Wikipedia, 22 10 2014. [Online]. Available: http://en.wikipedia.org/wiki/Inertial_navigation_system. [Accessed 23 10 2014].

[5] “Accelerometer,” Wikipedia, 13 10 2014. [Online]. Available: http://en.wikipedia.org/wiki/Accelerometer. [Accessed 23 10 2014].

[6] “Sensor Fusion on Android Devices: A Revolution in Motion Processing,” Google Tech Talk, 2 08 2010. [Online]. Available: http://www.youtube.com/watch?v=C7JQ7Rpwn2k. [Accessed 29 10 2014].

[7] “Accelerometer, Gyro and IMU Buying Guide,” [Online]. Available: https://www.sparkfun.com/pages/accel_gyro_guide. [Accessed 15 08 2014].

[8] “I2C,” Wikipedia, 27 10 2014. [Online]. Available: http://en.wikipedia.org/wiki/I%C2%B2C. [Accessed 29 10 2014].

[9] “IIC Bus,” [Online]. Available: http://www.i2c-bus.org/twi-bus/. [Accessed 29 10 2014].

[10] “Serial Peripheral Interface Bus,” Wikipedia, 22 10 2014. [Online]. Available: http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus. [Accessed 29 10 2014].

[11] “Triple Axis Accelerometer Breakout - LIS331,” Little Bird Electronics , [Online]. Available: http://littlebirdelectronics.com.au/products/triple-axis-accelerometer-breakout-lis331. [Accessed 1 08 2014].

[12] “LIS331HH,” Sparkfun, [Online]. Available: https://www.sparkfun.com/datasheets/Sensors/Accelerometer/LIS331HH.pdf. [Accessed 1 08 2014].

[13] “Triple Axis Accelerometer Breakout - MMA8452Q,” Little Bird Electronics, [Online]. Available: http://littlebirdelectronics.com.au/products/triple-axis-accelerometer-breakout-mma8452q-1. [Accessed 1 08 2014].

[14] “Xtrinsic MMA8452Q 3-Axis,12-bit/8-bit Digital Accelerometer,” Little Bird Electronics, [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/Accelerometers/MMA8452Q-

rev8.1.pdf. [Accessed 1 08 2014].

[15] “Triple Axis Accelerometer Breakout - ADXL345,” Little Bird Electronics, [Online]. Available: http://littlebirdelectronics.com.au/products/triple-axis-accelerometer-breakout-adxl345. [Accessed 1 08 2014].

[16] “ADXL345,” Analog Desvices , [Online]. Available: https://www.sparkfun.com/datasheets/Sensors/Accelerometer/ADXL345.pdf. [Accessed 1 08 2014].

[17] “9 Degrees of Freedom - Razor IMU - AHRS compatible,” Little Bird Electronics, [Online]. Available: http://littlebirdelectronics.com.au/products/9-degrees-of-freedom-razor-imu-ahrs-compatible-1. [Accessed 1 08 2014].

[18] “ 9 Degrees of Freedom - MPU-9150 Breakout,” Little Bird Electronics, [Online]. Available: http://littlebirdelectronics.com.au/products/9-degrees-of-freedom-mpu-9150-breakout. [Accessed 1 08 2014].

[19] “MPU-9150 Product Specification Revision 4.0,” Inven Sense, 14 5 2012. [Online]. Available: http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/IMU/PS-MPU-9150A.pdf. [Accessed 1 08 2014].

[20] “MinIMU-9 v3 Gyro, Accelerometer, and Compass (L3GD20H and LSM303D Carr,” Little Bird Electronics, [Online]. Available: http://littlebirdelectronics.com.au/products/minimu-9-v3-gyro-accelerometer-and-compass-l3gd20h-and-lsm303d-carrier. [Accessed 1 08 2014].

[21] “LSM303D,” Pololu, [Online]. Available: http://www.pololu.com/file/0J703/LSM303D.pdf. [Accessed 1 08 2014].

[22] “Arduino Due,” Sparkfun, [Online]. Available: https://www.sparkfun.com/products/11589. [Accessed 31 10 2014].

[23] “Arduino Mega 2560 R3,” Sparkfun, [Online]. Available: https://www.sparkfun.com/products/11061. [Accessed 31 10 2014].

[24] “Arduino/Processing Language Comparison,” Arduino, [Online]. Available: http://arduino.cc/en/Reference/Comparison?from=Main.ComparisonProcessing. [Accessed 31 10 2014].

[25] “MPU9150 Product Specification Revision 4.0,” [Online]. Available: http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/IMU/PS-MPU-9150A.pdf. [Accessed 08 2014].

[26] “What is Arduino?,” Arduino, [Online]. Available: http://arduino.cc/en/Guide/Introduction. [Accessed 23 9 2014].

[27] “C/C++ Comparison,” wiki, [Online]. Available: http://wiki.wiring.co/wiki/C/C%2B%2B_Comparison. [Accessed 23 9 2014].

[28] “Arduino Due,” Arduino, [Online]. Available: http://arduino.cc/en/Main/ArduinoBoardDue. [Accessed 31 10 2014].

[29] “Wire Library,” Arduino, [Online]. Available: http://arduino.cc/en/reference/wire. [Accessed 10 9 2014].

[30] “256K I2C™ CMOS Serial EEPROM,” [Online]. Available: https://www.sparkfun.com/datasheets/IC/24LC256.pdf. [Accessed 8 2014].

[31] “I2C/TWI LCD1602 Module (SKU: DFR0063),” wiki, [Online]. Available: http://www.dfrobot.com/wiki/index.php?title=I2C/TWI_LCD1602_Module_(SKU:_DFR0063). [Accessed 15 9 2014].

[32] “Download the Arduino Software,” Arduino, [Online]. Available: http://arduino.cc/en/Main/Software. [Accessed 23 08 2014].

[33] “Bi-Directional Logic Level Converter Hookup Guide,” Sparkfun, [Online]. Available: https://learn.sparkfun.com/tutorials/bi-directional-logic-level-converter-hookup-guide. [Accessed 25 09 2014].

[34] “I2C/TWI LCD1602 MODULE,” Little Bird Electronics, [Online]. Available: http://littlebirdelectronics.com.au/products/i2ctwi-lcd1602-module. [Accessed 16 9 2014].

[35] “Button,” Arduino, [Online]. Available: http://arduino.cc/en/tutorial/button. [Accessed 3 10 2014].

[36] [Online]. Available: http://www.i2cdevlib.com/usage. [Accessed 10 10 2014].

[37] GitHub, [Online]. Available: https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library. [Accessed 10 10 2014].

[38] GitHub, [Online]. Available: https://github.com/sparkfun/MPU-9150_Breakout/tree/master/firmware. [Accessed 10 10 2014].

[39] “float,” Arduino, [Online]. Available: http://arduino.cc/en/Reference/Float. [Accessed 17 11 2014].

[40] “double,” Arduino, [Online]. Available: http://arduino.cc/en/Reference/Double. [Accessed 17 11 2014].

[41] “Tutorial: Kalman Filter with MATLAB example part1,” youtube, [Online]. Available: http://www.youtube.com/watch?v=FkCT_LV9Syk. [Accessed 15 10 2014].

[42] “Xtrinsic MMA8452Q 3-Axis,,” Little Bird Electronics, [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/Accelerometers/MMA8452Q-rev8.1.pdf. [Accessed 1 08 2014].

[43] “I2C EEPROM - 256KBIT,” Little Bird Electronics, [Online]. Available: http://littlebirdelectronics.com.au/products/i2c-eeprom-256kbit. [Accessed 5 9 2014].

[44] “LOGIC LEVEL CONVERTER BI-DIRECTIONAL,” Little Bird Electronics, [Online]. Available: http://littlebirdelectronics.com.au/products/logic-level-converter-bi-directional. [Accessed 16 9 2014].

[45] “Wise Time With Arduino,” [Online]. Available: http://timewitharduino.blogspot.com.au/2009/05/getting-arduino-to-write-to-or-read.html. [Accessed 17 9 2014].

[46] “Binary Numbers Overview,” Help with PCs, [Online]. Available: http://www.helpwithpcs.com/hardware/binary-numbers.php. [Accessed 21 9 2014].

[47] “HD44780U (LCD-II),” [Online]. Available: http://www.dfrobot.com/image/data/DFR0009/HD44780.pdf. [Accessed 8 2014].

[1] “Global Positioning System,” Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Global_Positioning_System. [Accessed 15 12 2014].

[2] “Microelectromechanical systems,” Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Microelectromechanical_systems. [Accessed 15 12 2014].

[3] “Dead Reckoning,” 12 07 2014. [Online]. Available: http://en.wikipedia.org/wiki/Dead_reckoning. [Accessed 21 7 2014].

[4] “Inertial Navigation System,” Wikipedia, 22 10 2014. [Online]. Available: http://en.wikipedia.org/wiki/Inertial_navigation_system. [Accessed 23 10 2014].

[5] “Accelerometer,” Wikipedia, 13 10 2014. [Online]. Available: http://en.wikipedia.org/wiki/Accelerometer. [Accessed 23 10 2014].

[6] “Sensor Fusion on Android Devices: A Revolution in Motion Processing,” Google Tech Talk, 2 08 2010. [Online]. Available: http://www.youtube.com/watch?v=C7JQ7Rpwn2k. [Accessed 29 10 2014].

[7] “Accelerometer, Gyro and IMU Buying Guide,” [Online]. Available:

https://www.sparkfun.com/pages/accel_gyro_guide. [Accessed 15 08 2014].

[8] “I2C,” Wikipedia, 27 10 2014. [Online]. Available: http://en.wikipedia.org/wiki/I%C2%B2C. [Accessed 29 10 2014].

[9] “IIC Bus,” [Online]. Available: http://www.i2c-bus.org/twi-bus/. [Accessed 29 10 2014].

[10] “Serial Peripheral Interface Bus,” Wikipedia, 22 10 2014. [Online]. Available: http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus. [Accessed 29 10 2014].

[11] “Triple Axis Accelerometer Breakout - LIS331,” Little Bird Electronics , [Online]. Available: http://littlebirdelectronics.com.au/products/triple-axis-accelerometer-breakout-lis331. [Accessed 1 08 2014].

[12] “LIS331HH,” Sparkfun, [Online]. Available: https://www.sparkfun.com/datasheets/Sensors/Accelerometer/LIS331HH.pdf. [Accessed 1 08 2014].

[13] “Triple Axis Accelerometer Breakout - MMA8452Q,” Little Bird Electronics, [Online]. Available: http://littlebirdelectronics.com.au/products/triple-axis-accelerometer-breakout-mma8452q-1. [Accessed 1 08 2014].

[14] “Xtrinsic MMA8452Q 3-Axis,12-bit/8-bit Digital Accelerometer,” Little Bird Electronics, [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/Accelerometers/MMA8452Q-rev8.1.pdf. [Accessed 1 08 2014].

[15] “Triple Axis Accelerometer Breakout - ADXL345,” Little Bird Electronics, [Online]. Available: http://littlebirdelectronics.com.au/products/triple-axis-accelerometer-breakout-adxl345. [Accessed 1 08 2014].

[16] “ADXL345,” Analog Desvices , [Online]. Available: https://www.sparkfun.com/datasheets/Sensors/Accelerometer/ADXL345.pdf. [Accessed 1 08 2014].

[17] “9 Degrees of Freedom - Razor IMU - AHRS compatible,” Little Bird Electronics, [Online]. Available: http://littlebirdelectronics.com.au/products/9-degrees-of-freedom-razor-imu-ahrs-compatible-1. [Accessed 1 08 2014].

[18] “ 9 Degrees of Freedom - MPU-9150 Breakout,” Little Bird Electronics, [Online]. Available: http://littlebirdelectronics.com.au/products/9-degrees-of-freedom-mpu-9150-breakout. [Accessed 1 08 2014].

[19] “MPU-9150 Product Specification Revision 4.0,” Inven Sense, 14 5 2012. [Online]. Available: http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/IMU/PS-MPU-9150A.pdf. [Accessed 1 08 2014].

[20] “MinIMU-9 v3 Gyro, Accelerometer, and Compass (L3GD20H and LSM303D Carr,” Little Bird Electronics, [Online]. Available: http://littlebirdelectronics.com.au/products/minimu-9-v3-gyro-accelerometer-and-compass-l3gd20h-and-lsm303d-carrier. [Accessed 1 08 2014].

[21] “LSM303D,” Pololu, [Online]. Available: http://www.pololu.com/file/0J703/LSM303D.pdf. [Accessed 1 08 2014].

[22] “Arduino Due,” Sparkfun, [Online]. Available: https://www.sparkfun.com/products/11589. [Accessed 31 10 2014].

[23] “Arduino Mega 2560 R3,” Sparkfun, [Online]. Available: https://www.sparkfun.com/products/11061. [Accessed 31 10 2014].

[24] “Arduino/Processing Language Comparison,” Arduino, [Online]. Available: http://arduino.cc/en/Reference/Comparison?from=Main.ComparisonProcessing. [Accessed 31 10 2014].

[25] “MPU9150 Product Specification Revision 4.0,” [Online]. Available: http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/IMU/PS-MPU-9150A.pdf. [Accessed 08 2014].

[26] “What is Arduino?,” Arduino, [Online]. Available: http://arduino.cc/en/Guide/Introduction. [Accessed 23 9 2014].

[27] “C/C++ Comparison,” wiki, [Online]. Available: http://wiki.wiring.co/wiki/C/C%2B%2B_Comparison. [Accessed 23 9 2014].

[28] “Arduino Due,” Arduino, [Online]. Available: http://arduino.cc/en/Main/ArduinoBoardDue. [Accessed 31 10 2014].

[29] “Wire Library,” Arduino, [Online]. Available: http://arduino.cc/en/reference/wire. [Accessed 10 9 2014].

[30] “256K I2C™ CMOS Serial EEPROM,” [Online]. Available: https://www.sparkfun.com/datasheets/IC/24LC256.pdf. [Accessed 8 2014].

[31] “I2C/TWI LCD1602 Module (SKU: DFR0063),” wiki, [Online]. Available: http://www.dfrobot.com/wiki/index.php?title=I2C/TWI_LCD1602_Module_(SKU:_DFR0063). [Accessed 15 9 2014].

[32] “Download the Arduino Software,” Arduino, [Online]. Available: http://arduino.cc/en/Main/Software. [Accessed 23 08 2014].

[33] “Bi-Directional Logic Level Converter Hookup Guide,” Sparkfun, [Online]. Available: https://learn.sparkfun.com/tutorials/bi-directional-logic-level-converter-hookup-guide. [Accessed 25 09 2014].

[34] “I2C/TWI LCD1602 MODULE,” Little Bird Electronics, [Online]. Available: http://littlebirdelectronics.com.au/products/i2ctwi-lcd1602-module. [Accessed 16 9 2014].

[35] “Button,” Arduino, [Online]. Available: http://arduino.cc/en/tutorial/button. [Accessed 3 10 2014].

[36] [Online]. Available: http://www.i2cdevlib.com/usage. [Accessed 10 10 2014].

[37] GitHub, [Online]. Available: https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library. [Accessed 10 10 2014].

[38] GitHub, [Online]. Available: https://github.com/sparkfun/MPU-9150_Breakout/tree/master/firmware. [Accessed 10 10 2014].

[39] “float,” Arduino, [Online]. Available: http://arduino.cc/en/Reference/Float. [Accessed 17 11 2014].

[40] “double,” Arduino, [Online]. Available: http://arduino.cc/en/Reference/Double. [Accessed 17 11 2014].

[41] “Tutorial: Kalman Filter with MATLAB example part1,” youtube, [Online]. Available: http://www.youtube.com/watch?v=FkCT_LV9Syk. [Accessed 15 10 2014].

[42] “Xtrinsic MMA8452Q 3-Axis,,” Little Bird Electronics, [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/Accelerometers/MMA8452Q-rev8.1.pdf. [Accessed 1 08 2014].

[43] “I2C EEPROM - 256KBIT,” Little Bird Electronics, [Online]. Available: http://littlebirdelectronics.com.au/products/i2c-eeprom-256kbit. [Accessed 5 9 2014].

[44] “LOGIC LEVEL CONVERTER BI-DIRECTIONAL,” Little Bird Electronics, [Online]. Available: http://littlebirdelectronics.com.au/products/logic-level-converter-bi-directional. [Accessed 16 9 2014].

[45] “Wise Time With Arduino,” [Online]. Available: http://timewitharduino.blogspot.com.au/2009/05/getting-arduino-to-write-to-or-read.html. [Accessed 17 9 2014].

[46] “Binary Numbers Overview,” Help with PCs, [Online]. Available: http://www.helpwithpcs.com/hardware/binary-numbers.php. [Accessed 21 9 2014].

[47] “HD44780U (LCD-II),” [Online]. Available: http://www.dfrobot.com/image/data/DFR0009/HD44780.pdf. [Accessed 8 2014].